469

APPENDIX A

Contents

CHAPTER 1

Overview

Introduction

Atoma is a suite of development and management tools that enable effective administration of an organization's mobile computing fleet. Atoma features web-based deployment of applications and data to mobile clients, support for both connected and disconnected applications, remote monitoring and configuration of mobile devices and a highly extenbale and open architecture that will allows administrators and developers to utilize a wide range of technologies to completely integrate Atoma into an enterprise mobile solution.

Atoma consists of a set of server based tools and web services that allow administrators to configure Atoma servers, customize system operation and manage all mobile devices. A web-based application, known as the Console, is the central application used by an administrator to accomplish these functions. The Atoma server can be installed on servers running the AIX, Solaris or Windows 2000 operating systems.

In addition to the server components, Atoma includes a set of client programming objects and applications known as the Device Frameworks. The Device Frameworks provide application developers with programming extensions that aid in the creation of applications targeting multiple devices while speeding the overall development process. The Device Frameworks also include client applications that allow device users to utilize the services provided by Atoma and others that facilitate the operation of the overall system. Information on the Device Frameworks and application development is available in the **Atoma Developer's Guide**.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/)

In This Chapter

Server Architecture

There are several components that comprise the Atoma server. At the heart of a Atoma server installation is a Servlet Engine. The installation script for the server includes a Servlet Engine, however, if your organization has an existing Servlet Engine, it can be utilized instead of the engine included in the installation script. The Servlet Engine performs several functions including publishing the server's web services and housing the Console Application and its associated tools.

Due to the potentially large numbers of devices that a given server may support and the need to keep various items of information regarding each device, the server also includes a database. The server installation script does not include any database management system and thus allows you to utilize an existing database management system deployed in your organization. The server uses Java Database Connectivity (JDBC) to communicate with the database allowing a wide range of database management systems to be used.

Communications between server and mobile clients as well as inter-server communication is achieved via HyperText Transfer Protocol (HTTP). As a result of this the server publishes a number of Web Services for mobile clients designed to support client devices that are semi-connected/disconnected (i.e. devices that have periodic or intermittent network access) as well as devices that are always connected. These Web Services are used heavily in a number of the server processes described in the next section.

The server includes a Certificate Authority that is used internally by the system to manage digital certificates if certificate services are enabled. The Certificate Authority is embedded into the Atoma server such that it requires minimal administration. Certificates are associated with mobile clients registered in the system and the removal of any registered client automatically triggers the revocation of the client's certificate if applicable.

There are several components that comprise the Atoma server. At the heart of a Atoma server installation is a Servlet Engine. The installation script for the server includes a Servlet Engine, however, if your organization has an existing Servlet Engine, it can be utilized instead of the engine included in the installation script. The Servlet Engine performs several functions including publishing the server's web services and housing the Console Application and its associated tools.

Due to the potentially large numbers of devices that a given server may support and the need to keep various items of information regarding each device, the server also includes a database. The server installation script does not include any database management system and thus allows you to utilize an existing database management system deployed in your organization. The server uses Java Database Connectivity (JDBC) to

communicate with the database allowing a wide range of database management systems to be used.

Communications between server and mobile clients as well as inter-server communication is achieved via HyperText Transfer Protocol (HTTP). As a result of this the server publishes a number of Web Services for mobile clients designed to support client devices that are semi-connected/disconnected (i.e. devices that have periodic or intermittent network access) as well as devices that are always connected. These Web Services are used heavily in a number of the server processes described in the next section.

The server includes a Certificate Authority that is used internally by the system to manage digital certificates if certificate services are enabled. The Certificate Authority is embedded into the Atoma server such that it requires minimal administration. Certificates are associated with mobile clients registered in the system and the removal of any registered client automatically triggers the revocation of the client's certificate if applicable.

@

CHAPTER 2

System Installation

The system is installed via an installation script that guides you through the installation process. In order for the installation to be successful a number of requirements must be satisfied and the Pre-Installation tasks must be completed. The Pre-Installation steps must be performed so that the installation script can connect to the required systems/applications to verify correct configurations. If the installation script is unable to verify these configurations the product cannot be installed.

Installation Requirements

System Hardware Requirements

Server grade computer

300 Mhz or greater processor

100 MB Free Hard Disk Space

256 MB RAM

High speed network interface card

System Software Requirements

One of the following operating systems
- Microsoft Windows 2000
- IBM AIX 4.3
- Sun Solaris 2.6, 2.7 or 2.8

HTML web browser
- Browser does not have to be installed on the server machine but should be on any machine used to administer the server (recommended browsers: Microsoft Internet Explorer, Opera)

Database Requirements

One of the following Database Management Systems
- SQL Server 7.0 or 2000
- Oracle 8i
- DB2 7.1 or 7.2

JDBC type 2 or higher driver compatible with chosen datbase management system

User Management Requirements

One of the following User Management Systems
- IBM Secure Way Directory Server
- Microsoft Active Directory
- Microsoft Security and Accounts Manager (NT Domains)
- Netscape Directory Services
- SAP R/3
- Open LDAP Directory Server
- A system supporting the Lightweight Directory Access Protocol (LDAP)

To perform a system Installation:

Start the **Setup** executable for the appropriate system platform.
On IBM AIXÔ and Sun SolarisÔ systems this will be a file called *Setup.* On Microsoft WindowsÔ systems this will be a file called *Setup.exe*. The Welcome screen will be shown.

Click the **Next** button to advance to the next installation step.
If you wish to abort the installation click the **Cancel** button.

A message is displayed reminding you to perform the necessary Pre-Installation steps. If these steps have been performed, click the **Next** button to advance to the next installation step.
Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

Enter the pathname of the directory where you would like to install the software.
If the path entered does not exist you may be prompted to confirm that you would like the installer to create the necessary directories for you. Click **Yes** to have the directories created or **No** if you wish to create the directories onn your own. Click the **Next** button to advance to the next installation step. Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

Enter the host network address of the server where the system is being installed in the **Enter Host Name** input box.

This host network address will be used to construct the default URLs published by the server and to provide links to the Console (on page 17). The host network address may be an IP address (eg. *192.76.34.1*), a fully qualified domain name (eg. *myserver.mydomain.com*) or the system name of the server (eg. *myserver*).

Enter the TCP/IP port to be used by the server in the **Enter Port Number** input box.
The server will be configured to listen on the port entered and the port will be included in the default URLs published by the server and links created to the Console. The HTTP protocol is used for all communications with the server. If no port number is entered communications will be ordinarily routed to the default HTTP port (80) and must be routed from this port to the system using a web server or custom application. Click the **Next** button to advance to the next installation step. Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

Enter the appropriate JDBC driver class in the **Enter DB Driver** input box. (eg. *sun.jdbc.odbc.JdbcOdbcDriver*)
This driver will be used for all communications between system objects and the system's internal database. Consult your JDBC driver documentation to obtain the class name of the driver.

Enter the JDBC connection Uniform Resource Locator in the **Enter DB URL** input box. (eg. *jdbc:odbc:MyData*)
The string entered identifies a database server and database where the system tables will be stored. This database should be created on the database server prior to performing the system installation. Consult your JDBC driver documentation to obtain the proper format for the driver's connection URL.

Enter the database user username to be used in the **Enter User Name** input box.
This username will be used to authorize any connection made by the system to the system's internal database on the database server. The database user selected should have the rights to *create*, *update*, *insert*, *delete* and *drop* database objects on the system database.

Enter the password for the database user to be used in the **Enter Password** input box.

This password will be used to authenticate any connection opened to the system's internal database on the database server.

Re-Enter the password for the database user in the **Confirm Password** input box.

Enter the maximum number of connections that the system should make to the database server in the **Enter Maximum Pool Size** input box.
This value specifies the maximum number of concurrent connections that the system will open to the database on the database server. A value of -1 indicates that the system should not limit the number of concurrent connections that are opened to the database. The number entered may impact the performance of the system and is generally governed by the license agreement of the database management software being used. Consult your database documentation and license agreement if necessary.
Click the **Next** button to advance to the next installation step. Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

Select the appropriate a user authentication provider from the **Select LDAP Provider** drop down list.
The user authentication provider determines what type of user manager is used to validate users of the system. The system is capable of validating users against any LDAP compliant system and some non-LDAP systems.

Enter the host network address where for the user authentication provider in the **Host** input box.
This address determines where user authentication requests will be directed. When using an **SAP Directory Provider** the value entered should be the Host String of the application server where users should be validated. When using **Windows NT Authenciation** this value is not required and authentication request will be directed to the Windows Domain controller in the Windows Domain where the server is located.

Enter the port number where for the user authentication provider in the **Port** input box.
This number specifies the network port where user authentication requests will be directed at the user authentication server. When using an **SAP Directory Provider** the value entered should be the *System Number* of the application server where users should

be validated. When using **Windows NT Authenciation** this value is not required.

Enter the text that will appear above the Console login prompt in the **Enter Prompt Title** input box.
This text will appear in the web browser used to access the Console above the login input boxes.
Click the **Next** button to advance to the next installation step. The user authentication login box will appear. Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

Enter login information for the selected user authentication server to allow the installation to verify connectivity with the user authentication server. A valid username and password must be entered in the appropriate fields. When using an **SAP Directory Provider** a *Client* number must also be entered. When using **Windows NT Authenciation** or an **Active Directory Provider** a *Windows Domain* must be entered.
Click the **OK** button to continue.

Enter the name of the group of users whose members are authorized as adminstrators of the system in the **Enter the Administrators Group Name** input box. This does not apply when using an **SAP Directory Provider**.
A group should be created in the user management system of choice whose members are *administrators* of the system. These users will have access to the Console application for system administration.

Enter the name of the group of users whose members are authorized as users of the system in the **Enter the Users Group Name** input box. This does not apply when using an **SAP Directory Provider**.
A group should be created in the user management system of choice whose members are *users* of the system. These users will be allowed to authenticate the device setup process.

Enter the appropriate naming context under the user repository in the **Enter the Naming Context or select one from the list:** input box. (eg. *ou=MyOrganization, o=MyCompany, C=MyCountry*) This does not apply when using an **SAP Directory Provider** or **Windows NT Authentication**.
The *naming context* identifies the position in the user repository where the previously entered groups and the accounts for the group members can be found. If the installation script is able to retrieve the available contexts from the user

repository they will be displayed in a list below the input box for you selection. Click the **Next** button to advance to the next installation step. Click the **Back** button to return to the previous screen.If you wish to abort the installation click the **Cancel** button.

A summary of the installation options will be displayed for your review. Click the **Next** button to begin the installation processing. A screen with a progress bar will appear indicating the percentage of completion of installation processing.
Click the **Back** button to return to the previous screen. If you wish to abort the installation click the **Cancel** button.

When installation processing is complete the final installation screen is displayed.
A message will be displayed indicating the final status of the installation. If an error occured during the installation, the product will not be installed. The message displayed will indicate the location of a log file that contains more detailed information on the error. Once the error is resolved the installation must be restarted from the first step.

Click the **Finish** button to close the installer.

In This Chapter

Pre-Installation

Hardware Prerequisites

The system must be installed on a server grade computer equipped with a network interface card. The server must be equipped with one of the supported operating systems. The supported operating systems are Microsoft Windows 2000, IBM AIX 4.3, Sun Solaris 2.6 - 2.8. The system performs heavy hard disk read and write operations therefore a fast hard disk adapter should be installed on the server machine. The following server pre-requistes should be satisfied before installation:

1. The server's network interface card should be installed and operational.
2. If the system will be used in conjunction with a web server, the web server should be installed and operational.

Database Prerequisites

The system requires a database management system and a Java Database Connectivity (JDBC) driver. The supported database management systems are Micorsoft SQL ServerÔ, IBM DB2Ô, and Oracale 8iÔ Database server. The database management system used should be licensed for a minimum of ten (10) connections that can be exclusively used by the system. Before the system can be installed the following prerequisites must be satisfied:

1. Database management system must be installed. The machine where the system will be installed must have network access and/or a communication channel to the database management system.
2. A database must be created on the database management system. The system should have exclusive use of this database.
3. If desired a database user can be created for the exclusive use of the system.
4. A JDBC driver compatible with the database must be installed on the machine where the system is installed. The JDBC driver should be inserted in the classpath of this machine prior to system installation.

User Management Prerequisites

Users of the system are required to provide logon information before they are allowed to access the system. Any logon information entered by a user is authenticated aganst the user management system selected during installation. The supported user management systems are IBM Secure Way Directory Server, Microsoft Active Directory, Microsoft Security and Accounts Manager (NT Domains), Netscape Directory Services, SAP R/3, Open LDAP Directory Server and systems supporting the Lightweight Directory Access Protocol (LDAP). The foolwoing user management prerequisites must be satisfied before installation :

1. The user management system should be installed and operational. The server where the system will be installed should have network access and/or a communications channel to the user management system.
2. Desired users of the system should have valid inter-networking accounts on the user management system.
3. A group of users should be created for the users authorized as administrators of the system. The appropriate user accounts should be designated as members of this group.
4. A group of users should be created for the users authorized as device users in the system. The appropriate user accounts should be designated as members of this group.

Java Prerequisites

The components comprising the server software are all Java based. A number of third party Java products are installed along with the server software components :

1. **Java Development Kit**
On Windows platforms the Sun JDK 1.3.1 is installed
On IBM AIX platforms the IBM JDK 1.3.0 is installed
On Sun Solaris platforms the Sun JDK 1.3.1 is installed
2. **Apache Tomcat 3.2.3 Servlet Engine**

3. The following libraries are installed
Xerces 1.4.1 (xerces.jar)
Apache Soap 2.2 (soap.jar)
Java Cyptograpy Extensions 1.2.1 (jce1_2_1.jar, local_policy.jar, sunjce_provider.jar, US_export_policy.jar)
Java Secure Sockets Extensions 1.0.2 (jcert.jar, jnet.jar, jsse.jar)
Java Mail 1.2 (imap.jar, mail.jar, mailapi.jar, pop3.jar, smtp.jar)
Java Activation Framework 1.0.1 (activation.jar)
If a different version of these libraries is currently used on the server where the system will be installed and is on the classpath, it may be neccesary to remove these libraries prior to installation.

CHAPTER 3

Console

The Atoma **Console** is a browser based application that serves as the central administration interface for an Atoma system. Virtually all of the system's configuration options are accessble through the Console. Most changes made through the Console are instantly activated and do not require a stop and restart of the Atoma Server.

Console Login and Logout

A Console user must be authenticated before being allowed to access the application. When the Console is first opened a Login screen is displayed where valid login information must be entered before the Console application will be opened. The information entered will be validated against the user group designated as system administrators during installation of the system (see System Installation).

Once a user is validated, a Console session is created and the user may use the application. The session will timeout and be terminated after a period of inactivity. Once the session is terminated, the user will be required to login and create a new session before accessing the Console again. For information on setting the length of this period see Server Settings .

A **Logout** button/hot-spot is provided on each main screen of the Console. When this button is clicked the session is terminated. A user should logout before leaving the Console unattended to prevent unauthorized access.

Console Menu

The **Menu** is displayed at the left of the browser window when the Console is opened. Each item listed in the Menu is a hyper-link to a distinct module of the Console. Each module provides access to a set of related configuration options and/or allows execution of a set of related tasks.

Using the Console

After entering information and/or changing parameters through the Console, your changes must be activated. To activate changes an **Apply** button/hot-spot is provided on each module where changes can be made. After the **Apply** button is clicked, your changes become immediately effective.

To remove items from from your server settings and configurations a removal icon is ususally provided. When the removal icon is clicked, you are prompted to confirm the removal with a message box. If the removal is confirmed through the message box, the item is deleted and is effectively removed from the system. See the sections on Groups and Applications (on page 73) for more information on how to remove these items from your configuration.

Navigation between the options of the Console is achieved through the links that are provided on each screen displayed. A **Back** button/hot-spot is provided on many screens to allow you to return to the previous screen. While the **Back** button of your browser may also allow you to return to the previous screen in most cases, it is usually best to use the hot-spot provided on the screen.

In This Chapter

Devices

The **Devices** menu option provides access to Atoma's device management features. When a user perfoms the setup procedure against an Atoma server, a device registration is created for the device. The Devices menu option can be used to view the devices that are registered with a server and to perfrom a variety of management functions on each device.

Device Type List

The **Device Type List** is the default screen displayed when the **Devices** option is selected from the Console Menu. This screen displays the device type of every device supported by the installation and the following options related to device types:

`Search`	retrieves a list of devices matching search criteria and opens Device List View; see Searching for Devices
`Configure`	allows device configuration parmeters to be specified for the device type; see Setting Device Configuration
`Device Type`	the title of the device type category; when clicked performs a search for all registered devices matching selected device type; see Device Types & Searching for Devices
`Description`	a text description of the device type characteristics
`Processor`	the microprocessor family found in devices belonging to the device type
`Platform`	the operating system type and version found in devices belonging to the device type
`Model`	product number or name assigned by device manufacturer
`Manufacturer`	company that produces the device

`Auto Configure`	selection indicator specifying that auto-configuration information for this device type should be included when the auto-configure option is selected
`Auto configure selected device types`	allows the creation of auto-configuration information for all devices selected using the Auto Configure selection indicators; see Creating an Auto-Configuration

Device Types

Every device managed by the system is automatically assigned a device type. Device types can be thought of as hardware categories that are used to manage devices based on their hardware specifications and capabilities. The factors that determine the device type of a given device are microprocessor, operating system/platform, device manufacturer and model.

Generic Device Types

Generic Device Types are device types that are based solely on the microprocessor and operating system/platform used by the device. The system will automatically assign a generic device type when the device completes the setup process. Examples of generic device types are **PktPCSA1100, PktPCSH3**, and **PktPCR4000**. In the preceding examples each device type indicates that the device is a Windows Powered Pocket PCÔ and also indicates the type of processor present in the device.

Specific Device Types

Specific Device Types are device types that are based on the manufacturer and model of the device in addition to the microprocessor and operating system/platform. If a device is fully supported, the system will automatically assign a specific device type when the device completes the Tx-Sync process. If a device is not fully supported, the device will remained categorized under a generic device type even after the Tx-Sync process has been completed. Examples of specific device types are **CpqIpaq3xxx**, **IntPPC700**, and **CasPPCCpeia**. In the preceding examples each device type indicates the manufacturer and model of the device. The first is a *Compaq Ipaq 3000 series* device, the second is an *Intermec 700 series* device and the third is a *Casio Cassiopeia* device.

When a device is fully supported it will be assigned to a specific device type after it completes the Tx-Sync process for the first time. A device may remain categorized under a generic device type after completing the Tx-Sync process if (i) the Tx-Sync process was not completed successfully or (ii) the device is not fully supported.

If a device is not fully supported you may still be able to successfully manage the device using system however there are a few points to consider in this scenario:

- Device specific support is not available.
  Support for hardware peripherals such as scanners and model specific deployment are not possible.

- Device is untested.
  An unsupported device has not been tested with the device framework and the overall system. Therefore there can be no guarantee that all components of the system will function correctly and you assume all liablilty for any system failures that may occur.

Device Names

A **device name** is a unique identifier assigned to a device when the setup process is completed. The device name is entered by the user when the authentication screen is completed during the setup process. This name is checked for uniqueness during the setup process. If the name is not unique, the user will recieve an error and will be forced to choose another name. A device name may have a maximum of thirty-two (32) characters and may include any combination of alpha characters, numeric characters, spaces and underscores.

The purpose of the device name is to allow each device to be managed using a name that is familiar and/or has a significance for a system administrator. During the setup process, each device is assigned a unique identifier that is internal to the system (the device id). This device id is actually used by the system to refer to the device when necessary. This device id, however, is a global unique identifier (GUID) and the difference between two of these ids is not easily determined by the untrained eye. The device name allows a name of choice to be associated with each device such that each device is easily identified by the human eye when using the Console. (See Searching for Devices)

It is recommended that a system of naming devices be determined before deploying devices on the system

Searching for Devices

The **Search** option allows you to retrieve a device or list of devices that are currently registered on the system. There are two primary methods of obtaining this a list of devices: (i) by device type or (ii) by search string. After a search is executed, the Device List screen is displayed showing each device by device name and providing acces to a number of options related to each device.

Searching by Device Type

To retrieve a list of devices by device type, click on the desired device type name in the **Device Type List** screen. This will retrieve a list of every device registered on the system under the selected device type. The device type of a machine is automatically retrieved when the device completes the setup process or does a synchronization. A device will be listed under a generic device type immediately after it completes the setup process. However, if the device is supported, it will be listed under a specific device type after it completes a Tx-Sync.

Searching by Device Name

To retrieve a list of devices by search string:

Enter the search string in the input box next to the **Search** button.
The search string should be a character or sequence of characters that appear in the device name(s) of the device(s) that you wish to retrieve. It is not necessary to enter wild card characters such as "*" as they will be interpreted as part of the device name(s) that you are trying to retrieve. To retrieve a list of all devices, leave the search string input box empty.

Click on the **Search** button to execute your query.

Examples

Executing a search with the search string "*s*" will retrieve all devices with device names containing "*s*" such as: "*s*ale*s*100", "carlo*s*", or "ca*s*iodev2".

Executing a search with the search string "*tom*" will retrieve all devices with device names containing the sequence "*t-o-m*" such as: "*tom*cat-a", or "a*tom*all".

Device List

The **Device List** screen provides a listing of every device registered on the installation with a name that matches the search query entered. A device becomes registered on the installation after it successfully completes the setup process. The following information is displayed for each device:

`Removal`	check box used to delete device registration; see Removing Devices
`Device Specific Config`	allows device configuration parmeters to be specified for the selected device; see Setting Device Configuration
`Device Name`	"friendly" name of device; entered by device user during setup process;
`Device Type`	the device type of the hardware as currently identified by the system; see Device Types
`Inventory`	summary of applications installed on device; see Inventory
`Status`	information regarding operating condition of the device; see Status
`Log`	messages generated from server objects regarding selected device; see Log
`Sync Reset`	allows disposal of current synchronization for selected device; see Resetting a Device Sync
`Reset Data`	allows initialization of data piping process for selected device; see Resetting Device Data
`Group`	name of the Group used in the last succesfful synchronization by the device
`Last Updated`	date and time of the device's last successful synchronization

Setting Device Configuration

The **Configuration** options associated with devices and device types allow you to specify the operating system and hardware settings that you would like applied to a device. These settings include hardware settings such as *Ethernet IP address*, operating system settings such as *BackLight Brightness*, and common application settings such as *Browser Default Page*. Parameters specified using the Configuration options are applied to the appropriate device(s) during the first successfull synchronization, completed by the device, after the configuration parameters are saved using the Console.

The configuration parameters that can be modified using the Console are grouped into major categories. Within each major category, the parameters are grouped into a varying number of further minor categories and functional sections. The actual parameters that can be configured each belong to a functional section.

There are two methods of device configuration: **Device Specific Level Configuration** and **Device Type Level Configuration**. The same settings can be specified using either method and the two methods are designed to be used simultaneously.

Device Specific Level Configuration

Using the Device Specific Level Configuration method, the parameters desired are specified for each device on an individual basis. The settings specified in this manner are only applied to the single device selected when accessing the configuration options. In order to use Device Specific Level Configuration the desired functional section of parameters must be designated as **Device Specific**. Using this method it is possible to set a parameter to a value that will only be applied to a single specified device.

To modify device specific configuration settings

Locate the desired device using the Device List screen.
See Searching for Devices for information on locating devices.

Click on the **Device Specific Config** icon in the *Device Specific Config* column of the selected device.
The Configuration Parameters screen will be displayed

In each of the displayed functional sections change the available parameters as desired.

Click the **Apply** button/hot-spot near the bottom of the screen to save your changes. If you do not wish to save a change click the **Back** button/hot-spot to return to the Device List screen.

Device Type Level Configuration

Using the Device Type Level Configuration method, the parameters desired are specified for a selected device type. The settings specified in this manner are applied to each device identified as matching the selected device type. Using this method it is possible to set common parameters for multiple individual devices to the same value.

To modify device type configuration settings

Click on the **Configuration** icon in the *Configure* column of the selected device type.
The Configuration Parameters screen will be displayed

Click on the desired sections by clicking on the section name to view the available parameters under each functional section and edit the parameters as desired.
Click the **Apply** button/hot-spot on each section to commit your changes. If you do not wish to save a change do not click the Apply button and/or open another section by clicking on its name.

Click the **Back** button/hot-spot to return to the **Device Type List** screen.

Configuration Parameters

The following settings can be configured for a given device or device type:

`Battery - On Battery Power`	specifies how device behaves when device is powered only from the battery: **Timeout** - select a value from the list; value indicates length of time that device will stay on before suspending if not used.
`Battery - On External Power`	specifies how device behaves when device is powered externally: **Timeout** - select a value from the list; value indicates length of time that device will stay on before suspending if not used.
`Internet Explorer - Connection`	sets whether browser will attempt to automatically connect to the internet: **Auto Dial** - inidcates whether the browser automatically dial a connection when none is detected
`Internet Explorer - Settings`	general browser options **Play Sounds** - indcates whether sounds on pages will be downloaded and played **Fit to Screen** - indicates whether width of page content will be arranged to display without horizontal scrollbar **Default Page** - sets URL of browser home page **Show Address Bar** - inidcates whether URL address area will be displayed **Show Pictures** - indicates whether images on pages will be downloaded and shown **Warn Unsecure** - indicates whether a user alert will be inssued when transferring from secure to unsecure content
`Internet Explorer - History`	browser cache options **Days To Keep** - maximum age of an item in the history before it is deleted

`RAS Configuration - Connection`	specification of a Remote Access Service (RAS) phonebook entry on the device **Dial Timeout** - number of seconds to to wait for connection before cancelling **Country Code** - dialing code for country **Card Tone Wait Time** - time to wait or a Credit Card tone before continuing with the dialing process **Baud Rates** - indicates modem transmission speed **Domain** - windows networking domain to used for authentication User name - username/id to be used if required; if username is required and is left blank user will be prompted to enter username when entry is dialed **Wait for dial tone** - indicates whether RAS service should detect a dial tone before starting the dialing sequence **Modem** - the name of the modem to be used for the entry **Area Code** - dialing code for area **Phone Number** - telephone number to be dialed **Name** - text used to refer to the phonebook entry; appears in any listing of phonebook entries **Password** - user password to be used if required; if password is required and is left blank user will be prompted to enter password when entry is dialed Modem Commands -
`RAS Configuration - TCP/IP (advanced)`	basic TCP/IP settings of RAS phonebook entry **Slip** - indicates whether SLIP protocol will be used **Assignment** - specifies whether server will assign IP to device (*dynamic*) or device will specify own IP (*fixed*) **IP Header Compression** - indicates whether header compression will be used **IPAddress** - device IP if assignment is *fixed* **Use Software Compression** - indicates whether compression will be used
`RAS Configuration - Name Servers (advanced)`	TCP/IP name resolution settings of RAS phonebook entry **Secondary DNS address** - IP address of alternate DNS server **WINS address** - IP address of primary WINS server **DNS address** - IP address of primary DNS server **Assignment** - specifies whether server will assign Server IPs to device (*dynamic*) or a device will specify own Server IPs (*fixed*) **Secondary WINS address** - IP address of secondary WINS server

RAS Configuration - Port Settings (advanced)	communication settings of RAS phonebook entry **Data Bits** - number of bits used in data transmission **Flow Control** - specifies where transmission is controlled **Parity** - data transmission setting **Stop Bits** - number of bits used to specify transmission ends
Regional Settings	Specifies the *default language* and associated options used by device including *money* and *date format.* **Language** - indicates the region chosen for device formatting defaults
Owner Info	sets address and conctact information displayed as owner of device **Address** - owner's street address **Company** - organization to which owner belongs **Show On StartUp** - inidcates whether information will be shown automatically when device is turned on **Telephone** - owner's telephone number **Name** - owner's name **Email** - owner's internet email address
Owner Info - Notes	sets a text message that is displayed along with owner information **Show On StartUp** - inidcates whether information will be shown automatically when device is turned on **Message** - text information listed on the *Notes* tab of the *Owner Information* screen
Connectivity	options when connecting to a companion PC **Auto-connect serial** - specifies whether Active SyncÔ will automatically attempt to connect to a companion PC when a connection is detected
Backlight - On Battery Power	behavior settings when device is powered by battery **Turn on backlight when screen is tapped** - indicates whether screen backlight will be turned on when screen is tapped **Enable Turn off Backlight** - indicates whether operating system will automatically turn off backlight after a period of inactivity **Turn Off Backlight If not used for** - specifies period of inactivity length when operating system is set to automatically turn off backlight
Backlight - Brightness	specifies display settings (varies by device) **Enable Automatic Control** - indicates whether operating system will dim screen brightness based on power level **Level** - specifies desired brightness setting

`Backlight - On External Power`	behavior settings when device is powered externally Turn on backlight when screen is tapped - indicates whether screen backlight will be turned on when screen is tapped **Enable Turn off Backlight** - indicates whether operating system will automatically turn off backlight after a period of inactivity **Turn Off Backlight If not used for** - specifies length of period of inactivity when operating system is set to automatically turn off backlight
`Ethernet`	specifies LAN/WAN connection settings of device **Card Type** - name of ethernet driver to be configured
`Ethernet - TCP/IP`	specifies how device IP settings are assigned for ethernet driver **Subnet Mask** - mask to be used with address **Default Gateway** - address of gateway server **IP address** - address used by device if Assignment is *fixed* **Assignment** - specifies whether server will assign IP to device (*dynamic*) or device will specify own IP (*fixed*)
`Ethernet - Name Servers`	specifies name resolution settings for ethernet driver **Secondary DNS Address** - address of alternate DNS server **WINS Address** - address of primary WINS server **DNS address** - address of primary DNS server **Secondary WINS Address** - address of alternate WINS server

Creating an Auto-Configuration

The **Auto-Configuration** option allows you to create a configuration script(s) that can be stored on storage media such as Compact Flash cards or Secure Digital Cards and later executed on a client device. These configuration scripts will run automatically when the storage media is inserted into the device and will configure the RAS phonebook entry, network card settings and Internet Explorer settings of the device. The values configured by the script are determined by the currently active configuration settings (see Setting Device Configuration).

Two types of Auto-Configuration scripts can be created **Mulit-Platform Auto-Configuration** and **Single-Platform Auto-Configuration**. A Multi-Platform Auto-Configuration allows you to create mutiple scripts that are stored on a single storage card. When the card is inserted into the device, the corresponding script will automatically be launched. A Single-Platform script creates a single script that will only be launched when the card is inserted into a device of the appropriate platform. If any configuration values are designated as device-specific, the Single-Platform script will incorporate the configuration parameters set for the individual device if present.

To Create an Multi-Platform Auto-Configuration script

Click on the **Devices** menu option to open the **Device Type List** screen.
A list of all of the supported device platforms is displayed.

Use the **Configure** icon(s) to set the desired configuration parameters for each desired platform if necessary.
See Setting Device Configuration .

Place a checkmark in the **Auto-Configuration** checkbox adjacent to each desired platform.

Click the **Auto-Configure Selected Devices** button/hot-spot to create the scripts.
A screen displaying the selected devices is shown.

Enter a name for the Auto-Configuration script in the **Description** input box.
This name will only be used to identify the scripts when they are saved on the hard-drive of the machine.

Click the **Create Configuration** button to compile the Auto-Configuration script.

Copy the contents of the appropriate *Card* folder, inside the *DevicesConfig* folder, to the storage media.
The Auto-Configuration script(s) are saved to disk and are organized using a specific folder hierarchy. The script(s) are located in a subfolder of the system installation folder with the following format *[root installation folder] / DevicesConfig / [script-name] / Card*. For example, if the system is installed in a folder titled *Atoma* on the drive root and the name assigned to the script is *MyConfig*. The path to the configuration scripts would be */ Atoma / DevicesConfig / MyConfig / Card*. The entire contents of the *Card* subfolder should be copied to the storage media however the *Card* folder itself should not be copied.

To Create an Single-Platform Auto-Configuration script

Click on the **Devices** menu option to open the **Device Type List** screen.
A list of all of the supported device platforms is displayed.

Use the **Search** button display the desired device.
See Searching for Devices .

Click the **Device Specific Config** icon of the appropriate device.
The **Device Specific Configuration** screen will be displayed.

Change any of the displayed configuration parameters and click the **Apply** button/hot-spot if desired.
See Setting Device Configuration .

In the **Auto-Configuration** section below the configuration parameter list, click the **Configure** button/hot-spot.
A screen displaying the name of the selected device is shown.

Enter a name for the Auto-Configuration script in the **Description** input box.
This name will only be used to identify the scripts when they are saved on the hard-drive of the machine.

Click the **Create Configuration** button to compile the Auto-Configuration script.

Copy the contents of the appropriate *Card* folder, inside the *DevicesConfig* folder, to the storage media.
The Auto-Configuration script(s) are saved to disk and are organized using a specific folder hierarchy. The script(s) are located in a subfolder of the system installation folder with the following format *[root installation folder] / DevicesConfig / [script-name] / Card.* For example, if the system is installed in a folder titled *Atoma* on the drive root and the name assigned to the script is *MyConfig.* The path to the configuration scripts would be / *Atoma / DevicesConfig / MyConfig / Card.* The entire contents of the *Card* subfolder should be copied to the storage media however the *Card* folder itself should not be copied.

Inventory

The **Inventory** option allows you to view the applications that have been installed by the system on the selected device. The name and version of the application is displayed for each application installed on the device. The inventory information is updated during the Tx-Sync process and should usually match closely with the list of applications that have been assigned to the Group with which the device has syncrhonzied.

A common cause for a difference between the applications listed on a device's inventory and the applications assigned to the Group associated with the device is device type based application deployment. When an application is deployed for device type it will only be deployed to devices matching the chosen device type. An application of this type will not be present in the inventory of devices that do not match the targeted device type.

Status

The **Status** option allows you to view data regarding the operating condition of the selected device. The status data is recorded during the Tx-Sync process and reflects the state of the device as of the last completed Tx-Sync process. The categories that may appear on the status information screen are listed below:

`Memory`	displays the application memory status of the device **Load** - a percentage indicating how much of the total application memory is in use **Available** - the amount of application memory not in use **Total** - the total amount memory designated as application memory
`Storage`	displays the storage memory status of the device **Free** - the amount of memory available for file storage **Size** - the total amount of memory used for file storage
`Battery`	displays the power status of the device **Charge** - Overall battery condition **Status** - *Online* indicates device is powered externally; *Offline* indicates device is powered by battery **Life** - the percentage of charge remaining in the device's battery; *100%* indicates a full battery; *Unknown* indicates battery is being recharged **Chemistry** - type/construction of battery

Log

The **Log** option displays error messages that have been ouput by any server worker during a synchronization. If no errors have occurred while synchronizing the device this log is genrally empty. For each message the following is displayed: (i) **Event Time** - data and time that message was generated; (ii) **Subject** - general category of error; (iii) **Message** - error description.

Resetting a Device Sync

The **Sync Reset** option allows you to discard a device synchronization. Each synchronization performed by a device is assigned a unique identifier by the server. The server associates this unique identifier with any action performed during the course of the synchronization. The status of each server worker is monitored by the server as each synchronization is performed to determine the start and end of the sync. The server discards this status information once a synchronization is completed successfully. However, when a sync is aborted the server maintains the status of the sync to allow the process to be continued when an effort is made to resume the synchronization.

The Sync Reset option becomes available when a synchronization is started and remains until the syncrhonization completes successfully. This allows you to: (i) discard a synchonization during processing or (ii) discard a synchronization after it has been aborted due to error. It is generally not recommended to discard a synchronization during process unless it is absolutely necessary. It is possible that application data may be lost when a synchronization is discarded using the Sync Reset option depending on the state of completion of the sync.

When a synchronization is discarded all references to the synchronization are removed from the server and the next attempt by the device to synchronize will create a new synchronization, allowing the device to transmit a new set of application data and monitoring information to the server.

To reset device synchronization

Locate the desired device using the **Device List** screen. See Searching for Devices for information on locating devices.

Click on the **Reset** icon in the Sync Reset column of the selected device.
A message box will appear prompting you to confirm the disposal of the synchronization.

Click the **OK** button to discard the synchronization.
If you do not wish to discard the synchronization click the **Cancel** button.

CHAPTER 4

Resetting Device Data

The **Reset Data** option on the **Device List** screen allows you to re-initialize the Data Piping process for a specific device. When this option is applied, the client databases on the device will be destroyed. Consequently, instead of simply transmitting changes, the Data Piping process will send complete tables containing the most recent data to create a "clean" installation of the necessary databases on the device. This process occurs during the first synchonization of each device after the Reset Data option has been applied. Once this first synchronization is complete and the databases on a device have been re-created, the data piping process will resume normal operation where only the changes to the databases will be transmitted to the device.

To reset device data

Locate the desired device using the **Device List** screen. See Searching for Devices for information on locating devices.

Click on the **Reset Data** icon in the Reset Data column of the selected device.
A message box will appear prompting you to confirm the data reset operation.

Click the **OK** button to reset the data piping process for the device.
If you do not wish to reset the data piping process click the **Cancel** button.

Removing Devices

The **Removal** option allows you to delete device registrations from an installation. When a device is deleted all references to the device on the installation are removed and the device may no longer perform synchonizations with the server. If security has been enabled (see Server Settings), a deleted device may no longer communicate with any objects registered in the internal SOAP router (see SOAP Router Administration). Once deleted, a device is essentially forced to complete the setup process to create a new registration on the installation.

To Remove a Device

Locate the desired device using the **Device List** screen. See Searching for Devices for information on locating devices.

Place a checkmark in the checkbox adjacent to the device that is to be deleted.
You may delete multiple devices by placing a checkmark by each device to be deleted. To delete all devices on the installation place a checkmark in the **Remove All** checkbox in the title area of the list.

Click on the **Removal** icon below the list.
A message box will appear prompting you to confirm the deletion.

Click the **OK** button to confirm to delete the device.
If you do not wish to delete the device click the **Cancel** button.

Groups

A **Group** is a collection of applications. Groups are central to the application deployment and synchronization processes. When a client device synchronizes, the device user must select a Group with which the device will synchronize. The Group selected determines which applications are deployed to the device and correspondingly which database tables are created on the device if the Data Piping functionality is in use.

Groups List

The **Groups List** is the initial screen displayed when the Groups option is selected from the Menu. The Groups List displays all of the Groups that exist in the installation and provides the following information regarding each Group:

`Remove`	allows deletion of the inidicated group; see Removing a Group
`Name`	the title of the Group; see Displaying Group Details
`Applications`	the total number of applications assigned to the group; see Assigning Applications to a Group
`Data Preparation`	allows the Data Piping settings for packages associated with the Group to be modified; see Data Preparation Configuration
`Reset Data`	reinitializes the data stored on each device in the Group; see Resetting Group Data
`Groups Prompt Label`	text that appears on the device above the list of available Groups when the user performs a synchronization;

Adding a Group

The **Add Group** option allows you to create a new Group on an Atoma installtion. When a Group is created it will contain no applications and no associated devices.

To add a new Group:

Select the **Add Group** option from the **Group List** screen

Enter the title of the group in the **Name** input box
This name will be displayed to the device user in the list of available groups presented during synchronization. The name may consist of alphabetic characters, spaces and numbers. Two groups may not have the same name.

Enter the responsible party's e-mail address in the **Contact** input box
The responsible party will be notified via e-mail of synchronization errors if the Synchronization Settings are configured to send e-mails when errors occur.

Click on **Accept** to create the new Group.
If you no longer wish to add the Group, click the **Discard** hot-spot to return to the **Group List** screen.

Removing a Group

The **Remove** option allows you to delete a Group from an Atoma installation. When a Group is deleted it will no longer appear in the Group List screen of the Console and device users will no longer be able to synchronize with the deleted Group.

There can be no devices associated with the Group when a Group is deleted. A device is associated with a Group when the device user performs a successful synchronization using the Group. The association between the device and the Group is removed when the device user performs a successful synchronization against a different Group. To the Group that is associated with each registered device see Searching for Devices .

To remove a Group:

Click the Delete icon located to the left of the Group name on the **Group List** screen

A warning message will appear. Click **OK** to confirm that you wish to delete the Group.
If you do not wish to delete the Group click **Cancel**.

If no devices are associated with the Group it is deleted.
If devices are currently synchronized with the Group, the Group is not deleted and an error message is displayed.

Displaying Group Details

The **Group Detail** screen is displayed when the name of a Group is clicked on the **Group List** screen. The Group Detail screen displays the Group's current settings and allows these settings to be modified. The options available are as follows:

`Name`	displays the name of the group; this name will appear in a list on the device when the user performs a synchronization; this name may be changed by editing the value displayed;
`Contact`	the email address where system messages regarding the Group will be sent (if the email server option is activated); this address may be changed by editing the value displayed; see also Server Settings and Error Handling
`Applications section`	lists all applications registered on the installation; a checkmark adjacent to an application name indicates that the application is assigned to the group; see Assigning Applications to a Group
`Prompts section`	list the synchronization parameters associated with the Group; see Adding and Removing sync Parameters

Assigning Applications to a Group

The **Group Detail** screen displays all of the applications that have been created on the installation. Each application is listed with a checkbox indicating whether the application is currently assigned to the Group. An application can be assigned to a Group and removed from a Group using this screen as well as through the **Application Detail** screen (see Associating Groups with an Application).

To Assign an application to a Group

Open the **Group Detail** screen by clicking on the desired Group in the **Group List**.

Place a checkmark in the checkbox adjacent to the desired application
Checkmarks can be placed in more than one application to assign multiple applications to the Group. The application will be installed on devices synchronizing with the Group.

Click the **Apply** button/hot-spot to commit the changes.
If you do not wish to save the changes click the **Back** button/hot-spot.

To remove an application from a Group

Open the **Group Detail** screen by clicking on the desired Group in the **Group List**.

Remove the checkmark in the checkbox adjacent to the application to be removed
Checkmarks can be removed from more than one application to remove multiple applications from the Group. The application will be un-installed from devices synchronizing with the Group.

Click the **Apply** button/hot-spot to commit the changes.
If you do not wish to save the changes click the **Back** button/hot-spot.

Data Preparation Configuration

The **Data Preparation** option allows the Data Piping settings for a Group to be configured. It is important to note that the Data Piping settings that are configured using this option apply only to the Group that has been selected. When the Data Preparation icon is selected the **Group Packages List** is displayed.

Group Packages List

The **Group Packages List** screen displays each data piping package that is associated with the selected group. The data piping packages that are used by an application are defined using the Application Deployment tool. When an application is assigned to a Group, the packages used by the application are associated to the Group. If multiple applications in a Group use the same package, the package is only associated to the Group once and as a result it is only listed once in the Group Packages List. When the last application using a package is removed from the Group, the association between the package and Group is also removed.

The following options are available for each package displayed in the Group Packages List:

`Reset Data`	causes the client data table defined by the package to be re-initialized; see Resetting Package Data
`Schedule`	determines when the data piping process will be performed for the package; see Scheduling Data Preparation

Scheduling Data Preparation

The **Schedule** option allows you to determine when **Data Preparation** will occur for a package referenced by a Group. When data preparation occurs, the data source defined in the package is queried to retrieve the latest data set and the client device data set defined by the package is updated accordingly. The end result of the data preparation process is that the database changes necessary on each device that synchronizes with the Group are determined as defined by the data piping package. There are two options available to specify when this process occurs for a package in a Group:

To specify when Data Preparation will occur

Click the **Schedule** icon adjacent to the desired package in the **Group Packages List** screen.

Select one of the available scheduling options:

During Synchronization	specifies that data preparation for each device will occur during the Tx-Sync process; changes to data sets will be determined when device user executes the synchronize option
By Schedule	specifies that data preparation for all devices will occur at predetermined times during a day; see **Preparation By Schedule** below

Click the **Apply** button/hot-spot to commit the change. If you do not wish to change the scheduling option click the **Back** button/hot-spot.

Preparation By Schedule

When Data Prepartion occurs by schedule, the data set changes necessary on each device is determined at a specified time during the day. A day is defined as a twenty-four hour period beginning at 12:00 AM. Data preparation for the package will be executed for every device that last synchonized with the group at any time that is added to the schedule.

To add a time to the schedule

Click the **Schedule** icon adjacent to the desired package in the **Group Packages List** screen.

Select the hour portion of the desired time from the first drop-down list.

Select the minutes portion of the desired time from the second drop-down list.
The selections available from this drop-down list depend on the interval configured for the Data Piping Scheduler in the Server Settings section.
For example: if the Data Piping Scheduler is configured to operate at intervals of twenty (20) minutes, the options available in the list are :00, :20 and :40. If the scheduler interval is sixty (60) minutes the only option available is :00.

Select whether the time is in the **AM** or **PM** from the drop-down list.

Click the **Add** (@) icon to insert the time into the schedule.

To remove a time from the schedule

Click **Removal** icon adjacent to the time to be removed from the schedule.

The time is removed from the schedule.

Resetting Group Data

The **Reset Data** option on the **Group List** screen allows you to re-initialize the Data Piping process for each device that synchronizes with a Group. When this option is applied, the client databases on each device that synchronizes with the re-initialized Group will be destroyed. Consequently, instead of simply transmitting changes, the Data Piping process will send complete tables containing the most recent data to create a "clean" installation of the necessary databases on the device. This process occurs during the first synchonization of each device after the Reset Data option has been applied. Once this first synchronization is complete and the databases on a device have been re-created, the data piping process will resume normal operation where only the changes to the databases will be transmitted to the device.

To reset the data associated with a Group:

Click the Reset Data icon located to the right of the Group name on the **Group List** screen

A warning message will appear. Click **OK** to confirm that you wish to reset the data of the Group.
If you do not wish to reset the data click **Cancel**.

The Data Piping process is re-initialized for the Group.

Resetting Package Data

The **Reset Data** option on the **Group Package List** screen allows you to re-initialize the Data Piping process for a specific table on each device that synchronizes with a Group. After data has been reset for a data piping package, each device that synchronizes with the Group will recieve a complete table with the latest data set to replace the current client database table on the device defined by the package. The current client database table will be destoryed and recreated using the data provided in the latest data set. This process happens during the first synchonization of each device after the Reset Data option has been applied to the data piping package on the Group Package List screen. After the client data defined by the package has been completely re-initialized and the synchronization is complete, subsequent synchronizations will resume normal operation of the data piping process where only the changes to the data set defined by the package will be transmitted to the device.

To reset the data of a single package for a Group:

Click the Reset Data icon located to the right of the Package name on the **Group Packages List** screen

A warning message will appear. Click **OK** to confirm that you wish to reset the data of the Package. If you do not wish to reset the data click **Cancel**.

The Data Piping process is re-initialized for the the package in the Group.

Note that the Data Piping process handles the packages associated with each Group separately. If a data piping package is referenced in mulitple groups, applying the Reset Data option to the package in one group will not affect the package in another Group.

Adding and Removing sync Parameters

A **Tx-Sync Parameter** is a value that a device user is prompted to enter when the user starts the synchronization process from the device. Tx-Sync parameters are associated with Groups through the Group Detail screen. The prompts that appear on the device depend on the Group selected by the user for syncrhonization. A list of all of the Tx-Sync parameters (or prompts) associated with a group is displayed on the **Group Detail** screen.

To create a Tx-Sync Parameter

Open the **Group Detail** screen by clicking on the desired Group on the **Group List** screen.

Click the **Add Prompt** button/hot-spot below the list of current prompts.
If there are no prompts associted with the group the list of prompts will be empty.

Fill out the **Add Prompt** form as specified below:

Prompt Name	the name of the prompt; this name will be displayed to the user on the device; this is also the key used when accessing the value programatically from a Tx-Sync extension object
Prompt Type	specifies how the prompt will be presented to the user; (i) **TEXT** - prompt will allow user to enter information in an input box (ii) **PASSWORD** - prompt will allow user to enter value in an input box and value will be masked as it is entered (see also TxSync Parameters (on page 153))
Null	indicates whether value is required; (i) **No** - user must enter a value when prompted (ii) **Yes** - user may leave the value empty
Data Type	specifies what type of data can be entered by the user; (i) **String** - any character on keyboard can be entered (ii) **Integer** - value may consist of numbers and a leading negative sign if necessary.
Default	specifies a value that will be filled by default (see also TxSync Parameters (on page 153))

Max Length	specifies a maximum number of characters that can be entered; when left blank or set to zero the number of characters is not limited

Click the **Apply** button/hot-spot to create the new prompt.
If you do not wish to create the prompt click the **Back** button/hot-spot.

To remove a Tx-Sync Parameter

Open the **Group Detail** screen by clicking on the desired Group on the **Group List** screen.

Click on the **Removal** icon adjacent to the prompt that you wish to delete.
A message box will be shown asking you to confirm the deletion.

Click the **OK** button in the confimation message box.
If you do not wish to delete the prompt click the **Cancel** button.

CHAPTER 5

Applications

The **Applications** option provides access to the the complete list of device applications registered on the system. A device application is a set of files that has been packaged using the Application Deployment tool. Once the application is assigned to a Group, the files that make up the application will be deployed to any device that synchronizes with the Group. The options available from the **Application List** screen are as follows:

`Action`	allows removal and deletion of the selected application; see
`Name`	the title of the Group; see Displaying Application Details
`Version`	displays the version number assigned to the application; this value is assigned during Application Deployment
`Groups Assigned`	displays the total number of groups associated with this application
`Add Application`	registers an application from a zip file; see Adding an Application (on page 75)
`Application Deployment`	launches the Application Deployment tool

In This Chapter

Adding an Application

The **Add Application** option allows you to register a new device application on an Atoma installation. When the application is initially registered it will not be assigned to any Groups. An application must be assigned to a Group before it will be deployed to any client devices.

To add a new Application:

Select the **Add Application** option from the **Application List** screen

Click the **Browse** button/hot-spot next to the Zip File input box to select the application zip file.
You will be prompted to navigate to the location of the zip file on your local machine and to select the desired file. An application zip file can be created using the Application Deployment tool.

Click the **Apply** button after the name and path of the file are displayed in the Zip file input box.
The application files will be transferred to the server and the application will be registered on the installation. If you do not wish to add the application click the **Discard Changes** button/hot-spot.

The new application is displayed on the **Application List** screen.

Removing an Application

The **Action** option allows you to *uninstall* and *remove* an application from your installation. To delete an application from the system the first step is to *uninstall* the application and second is to *remove* the application. These two steps are enforced to prevent version control errors that would occur otherwise.

When an application is uninstalled the application is removed from any Groups to which it may be assigned and it is flagged for deletion from the system. Once flagged for deletion, the application cannot be assigned to a Group and the application will be removed from any client devices where it is installed as each device is synchronized with the server.

After the application has been uninstalled from each device where it was installed, through syncrhonication, the application can be *removed.* When an application is removed all references to the application are completely deleted from the installation. After an application is uninstalled there may be a period where the system prevents the application from being removed. This is because there are client devices, on which the application is installed, that have not yet synchronized and so the system has not had the opportunity to remove the applications from these devices. Once these devices have synchronized, the application can be successfully removed. To view the applications installed on a client device use the Inventory option on the **Device List** screen.

To Uninstall an Application:

Click the **Uninstall** icon adjacent to the application's name on the **Application List** screen.

Click the **OK** button when prompted to confirm that you wish to uninstall the application.
If you no longer wish to uninstall the application click the **Cancel** button.

The application is uninstalled and the **Removal** icon will replace the Uninstall icon on the left of the application name.

To Remove an Application:

After uninstalling the application, click the **Removal** icon adjacent to the application's name on the **Application List** screen.

Click the **OK** button when prompted to confirm that you wish to completely delete the application.

If you no longer wish to delete the application click the **Cancel** button.

The application is completely deleted from the installation and no longer appears on the **Application List** screen.

Displaying Application Details

The Application Detail screen displays all of the attributes of the selected application and allows some of the application's attributes to be modified. The following information, specified during Application Deployment , is displayed on the Application Detail screen and cannot be modified:

`Name`	the title of the application
`Description`	text that may provide a brief summary of the application
`Version`	displays the version number assigned to the application
`Installed Date`	displays the total number of groups associated with this application
`File To Execute`	lists the file that will be executed or loaded when the application is started
`Supported Device Types`	lists the device types that the application targets; if the generic type is included the application can be deployed to any type of device
`Files To Register`	lists the files that will be registered with the operating system when the application is deployed
`Referenced Data Piping Packages`	lists the Data Piping Packages that are associated with the application

The following attributes of an application can be modified from the Application Detail screen:

`Pre-Worker / Post-Worker Status`	inidcates if the application should be registered as a client worker; see Registering a Client Worker
`Groups`	displays the groups currently associated with the application; see Associating Groups with an Application

Registering a Client Worker

The Tx-Sync Agent will attempt to execute each application designated as a client worker during each synchronization. An application registered as a *pre-worker* will be executed before files are transmitted from the device to the server. An application registered as a *post-worker* will be executed after files have been received by the Tx-Sync Agent on the device. (see also Tx-Sync Agent and Client Workers)

In order for the client worker to be successfully executed, the client worker application must be installed on the device i.e. the client worker application must be assigned to the group with which the device last synchronized.

To register an application as a client worker

Open the **Application Detail** screen by clicking on the desired application in the **Application List**.

Place a checkmark in the **Pre-Worker** and/or **Post-Worker** checkbox
It is possible for one application to be registered as both a Pre-Worker and a Post Worker. In this case the application will be run twice during each synchronization.

Click the **Apply** button/hot-spot to commit the registration.
If you do not wish to save the changes click the **Back** button/hot-spot.

To un-register an application as a client worker

Open the **Application Detail** screen by clicking on the desired application in the **Application List**.

Remove the checkmark in the **Pre-Worker** and/or **Post-Worker** checkbox

Click the **Apply** button/hot-spot to commit the changes.
If you do not wish to save the changes click the **Back** button/hot-spot.

Associating Groups with an Application

The **Application Detail** screen displays all of the Groups that have been created on the installation along with the attributes of an individual application. An application can be assigned to a Group and removed from a Group using this screen as well as through the Group Detail screen (see Assigning Applications to a Group).

To add an application to a Group

Open the **Application Detail** screen by clicking on the desired application in the **Application List**.

Place a checkmark in the checkbox adjacent to the desired Group
Checkmarks can be placed in more than one Group to associate the application with multiple Groups. The application will be installed on devices synchronizing with the Group(s) specified.

Click the **Apply** button/hot-spot to commit the changes.
If you do not wish to save the changes click the **Back** button/hot-spot.

To remove an application from a Group

Open the **Application Detail** screen by clicking on the desired application in the **Application List**.

Remove the checkmark in the checkbox adjacent to the desired Group
Checkmarks can be removed from more than one Group to remove the application from multiple Groups. The application will be un-installed from devices synchronizing with the Group(s) specified.

Click the **Apply** button/hot-spot to commit the changes.
If you do not wish to save the changes click the **Back** button/hot-spot.

Application Deployment

The **Application Deployment** tool is used to register new applications and to create application archives that can be used to register applications using the Console. An application is essentially a group of related files that is automatically deployed to the device by the system.

The Application Deployment tool can be used to create two basic types of applications:
(i) a *traditional* application - an application that is designed to be launched by the user or by the Device Framework as a client worker
(ii) a *file-deployment* application - an application created with the tool solely for the purpose of distributing files to the device or installing external components or applications (e.g.CAB files)
The *main app* of an application (see *Main App To Execute* in **To Deploy a new application** below) is the file that begins the application. This file is launched or loaded to start the application. *Traditional* applications require that a *main app* be defined for the application while *file-deployment* applications do not require a *main app* to be defined. The file defined as the *main app* must be associated with all device types supported by the application.

The Application Deployment tool allows you to specify the device types that are supported by the new application. When an application supports multiple device types it is necessary to specify each file that is part of the application for each device type. You must specify each file that is to be copied or installed on the device for each device type. When the application is installed on a device of a supported device type, the only files installed on the device are the files that have been associated with the supported device type using the tool.

The Application Deployment tool allows you to specify support for all devices through the generic platform. The application files associated with the generic platform are installed on a device if the application does not include support for a device type that matches the device. If the application does include support for a device type that matches the device, the files associated with the device type are installed instead of the files associated with the generic platform.

To deploy a new application

Open the **Application Deployment** tool by clicking on it's name on the **Tools** menu.

Place a check mark in the checkbox adjacent to each device type that the application will support.
The **Generic** device type can be used to create an application that will be deployed to every device type other than the types specifically supported by

this application. For more information on device types see Device Types .

Click the **Next** button/hot-spot to continue with application deployment.
The **Application Registration** from will be displayed.

For each device type selected previously a file selection form is displayed. Use the **Browse** button to locate files on your local drives to be included in the application for the selected device type. Use the **Add** button to include the file in the application for the selected device type. The **Files Added** box displays the files that have been included in the application for each device type.
Note that the files added to a device type are the only files that are included in the application for that device type. If you want to include the same file in the application for multiple device types you must add the file to each device type using the forms displayed for each device type. A version of the file that will be designated as the Main App to Execute (see next step) must be included in all supported device types.

Complete the **Application Registration/Manifest** form.
The following information must be entered:

Name	the title of the application; this is the text that will appear in any list of applications that includes this application
Description	text that is shown to users to give additional information about the application
Version	indicators that can be used to visually identify a particular release of an application: **Major, Minor, Release, Version** - the values entered in each one of these fields will be appended in the above order to form the visual version identifier This version is not used by the system to perform version control.)
Icon	image file displayed when application is displayed on client device; any image file added to the application can be selected as the icon (supports image files of type GIF=.gif, BITMAP=.bmp and JPEG=.jpg)

App. Type	specifies how the main application starting point (below) is to be launched: (i) **ASP** - application is a Microsoft Acive Server Pages seb application (ii) **EXE** - application is a binary executable file (iii) **EVB** - application is an eMbedded Visual Basic file (iv) **COM** - application is a Microsoft Component Object Model object (v) **HTML** - application is an HTML page
Main App to Execute	specifies a file that is to be launched or loaded when the application is started by the user **ASP** - the complete URL of the applicaiton must be entered **EXE** - select the file to be launched from the list of executables included in the application **EVB** - select the file to be launched from the list of eMbedded Visual Basic files included in the application **COM** - the Program ID of the object must be entered **HTML** - select the file to be loaded from the list of HTML pages included in the application
Show App	indicator that specifes whether application will appear in list presented to device users when installed: **True** - application is shown in application list on client device; **False** - application is not shown in application list on client device
Data Packages	allows the data packages used by the application to be specified; a list of all data packages registered on the system is displayed - place a checkmark in the checkbox adjacent to any data packages used by the application
Files To Register / Execute	specifies the exectuables (EXEs) that will be launched and/or dynamic link libraries (DLLs) that will be registered when the application is installed on a client device; a list of DLLs and EXEs is displayed - place a checkmark in the checkbox adjacent to each file that you wish to launch or register when the application is installed;

Click the **Next** button/hot-spot to move to the next screen.
The final screen of the **Application Deployment** tool is shown. If you wish to return to the **Device Type Selection** screen click the **Back** button/hot-spot.

Click **Deploy Application** to immediately register the application on the installation or click **Download Zip File** to receive a ZIP archive containing all of the application files. A downloaded ZIP archive can be used to later register the application on an installation. See Adding an Application (on page 75).

If the **Deploy Application** option was selected the application is registered and you are returned to the **Tools** menu. If the **Download Zip File** option was selected your browser's standard download dialog will open. Select the save file option and save the file for later registration of the application.

CHAPTER 6

Settings

The **Settings** menu option provides access to a number of global parameters that affect the operation of the overall system. Each group of parameters is accompanied by an **Apply** button/hot-spot. The following options are available on the Settings screen:

`Client Settings`	configure global settings for client devices; see Client Settings
`Server Settings`	configure Tx-Sync process and server parameters; see Server Settings
`Security`	set the security options of the installation; see Security
`Load Balancing`	configure multi-server installationss
`Server Workers`	configure syncrhonization modules

In This Chapter

Client Settings

The **Client Settings** option allows you to set values that affect the operation of the Device Framework. These values apply to all devices that are registered on your installation. The following values can be set using the Client Settings option:

`Soap URL`	specifies the URL that will be used during the Tx-Sync process to to send and receive the required messages; the default value of this URL is constructed based on the host name entered during the system installation; it is generally only necessary to change the host portion of this URL
`Synchronization Timeout`	the length of time that the client device will wait between synchronization retries (see below); during the Tx-Sync process the client device waits for a period while the server completes synchronization tasks; this value helps determine the length of time that the client device will wait before reporting to the user that the sync has timed out.
`Synchronization Retries`	number of times that the client device will check to see if the server has completed the synchronization; if the server has not completed the synchronization the device will wait for a duration equal to the synchronization timeout period (see above) before again checking with the server; during the Tx-Sync process the client device waits for a period while the server completes synchronization tasks; this value helps determine the length of time that the client device will wait before reporting to the user that the sync has timed out.
`File URL`	specifies the URL that will be used during the Tx-Sync process to exchange files with a server; the default value of this URL is constructed based on the host name entered during the system installation; it is generally only necessary to change the host portion of this URL

To change a Client Setting:

Change the Client Setting parameter by editing the value in the input box next to the parameter name that you wish to change

Click the **Apply** button/hot-spot below the Client Settings section to save your changes.
Any changes made to the Client Settings will be applied to: (i) each device that completes the setup process after the changes are applied and (ii) each device that successfully synchronizes after the changes are applied.

Server Settings

The **Server Settings** option allows you to set the parameters that affect the overall server operation. Care should be exercised when setting these parameters because many of the parameters are instantly activated when changes are applied. Parameters that require the server to be restarted after they have been applied are noted where applicable. The following values can be configured:

`Error Handling during synchronization`	Defines the behavior of the server when an error occurs during the Tx-Sync process: (i) **ABORT** - the sychronization will be stopped; until the error is resolved the client device will be unable to synchronize successfully;(ii) **CONTINUE** - the syncrhonization will be completed (iii) **ABORT AND MAIL** - the sychronization will be stopped and an email will be sent to the person listed as the Group's contact; until the error is resolved the client device will be unable to synchronize successfully. (iv) **CONTINUE AND MAIL** - the syncronization will be completed and an email will be sent to the person listed as the Group's contact;
`Email Server`	the host name or address of the mail server to be used by the system to send emails
`Set Date and Time during synchronization`	specifies whether the clock of each client device should be sychronized with the server during the Tx-Sync process
`Data Preparation Schedule Interval`	period of time in minutes that the Data Piping Scheduler will pause between attempts to retrieve events from the Data Piping schedule. Note that the scheduler works from the top of the hour i.e. a value of 60 minutes causes the schedule to be checked at Noon, 1pm, 2pm etc.... a shedule of 20 minutes causes the schedule to be checked at Noon, 12:20pm, 12:40pm, etc...; after this value is applied the server must be restarted for the changes to take effect.
`Return sync estimate during syncrhonization`	specifies whether an estimate of the length of time to perform a syncrhonization is made available to the device

Maximum number of concurrent synchronizations	number of device syncrhonizations that can be processed concurrently by the server; when this maximum is reached additional syncrhonizations are queued before processing
Soap timeout	period of time in seconds that the Tx-Sync process will allow for each asynchronous SOAP transaction to be executed on behalf of the client device; after this value is applied the server must be restarted for the changes to take effect.

To change a Server Setting:

To select an option from a list, click the radio button adjacent to the option name.

To change the Server Setting parameter, edit the value in the input box next to the parameter name that you wish to change

Click the **Apply** button/hot-spot below the Server Settings section to save your changes.

Security

The **Security** section provides access to parameters that affect how the server handles incoming requests from client devices. In addition to the security features provided by the system, it is recommended that the SSL protocol (*https*) be utilized to ensure that communications between the device and server are protected. The following parameters can be modified:

`Security Status`	specifies whether incoming synchronization and SOAP requests should be challenged to determine if the requester has a digital certificate issued by the system; by default the value is false and incoming requests are not challenged
`Synchronization Guard`	allows a custom object to be specified for handling pre-synchronization tasks and security; see Synchronization Guard

To change a Security Setting:

To select an option from a list, click the radio button adjacent to the option name.

To change a parameter, edit the value in the input box next to the parameter name that you wish to change or select the value from the list provided

Click the **Apply** button/hot-spot below the Security section to save your changes.
Any changes made to the Security settings become immediately effective when applied.

Synchronization Guard Parameters:

`Server URL`	URL of the Web Service where the syncrhonization guard object is registered
`Object Name`	name of the Web Service that implements the syncrhonization guard object
`Soap Implementation`	indicates the envelope format used by the Web Service
`Enable Validation`	enables/disables use of the specified guard

Load Balancing

The **Load Balancing** option is used to configure the syncrhonization HTTP transport settings of the installation. HTTP transports are used by the server to transfer and receive large sets of data during the TxSync process. To increase the synchronization capacity of an installation, additional HTTP transports can be added allowing the server to distribute the transfer of large data accross multiple machines. Multiple HTTP transports allow the server to perform an increased number of simultaneous synchronizations while maintaining lower total syncrhonization times (duration of a single TxSync).

To Add an HTTP Transport

Use the Atoma installation script to install the HTTP Transport on the target server.

Open the **Load Balancing** screen by selecting on Settings->Load Balancing from the main menu.

Click the **Add** button/hot-spot.
An edit box for the transport URL is displayed.

Enter the full URL for the HTTP transport in the input box.
If the new transport is secured by SSL the protocol should be changed to https://.

Click the **Apply** button/hot-spot.
The new transport will be configured on the installation.

To remove an HTTP Transport

Open the **Load Balancing** screen by selecting on Settings->Load Balancing from the main menu.

Click the **Removal** icon adjacent to the URL to be removed.
A message box will appear prompting you to confirm the removal.

Click the **OK** button to confirm removal of the HTTP Transport.
If you do not wish to remove the transport click the **Cancel** button.

Server Workers

The **Server Workers** tool allows you to add a custom Server Worker to the Tx-Sync process. There are a number of system Server Workers that perform the basic tasks executed in all device synchronizations. By adding a custom Server Worker to the Tx-Sync process, you are able to include any task(s) desired as part of the synchronization process that occurs for each device . When a Server Worker is added to the Tx-Sync process it is executed during every synchronization performed by a device regardless of which Group is selected for the synchronization. However when developing a server worker it is very straightforward to filter the tasks performed based on the Group selected during synchronization. See Custom Server Workers for more information. The Custom Server Workers List is displayed when the Server Workers tool is opened and shows the following information:

Name	the full class name of the custom server workers
Has Dependencies	indicates whether the worker depends on another registered server worker

To Add a Custom Server Worker

Ensure that the Server Worker class and the classes used by the Server Worker are included on the classpath used by the server.
There are a number of ways of accomplishing this. One method is to add the jar(s) containing the desired classes to the "..\WEB-INF\lib" folder used by the Atoma web application folder. A restart of the server is usually necessary after performing this step.

Open the **Server Workers** screen by selecting on Settings->**Server Workers** from the main menu.

Click the **Add Workers** button/hot-spot.
The Add Worker form is displayed.

Enter the full class name of the Server Worker class in the **Class Name** input box.
This name consists of the package and class name of the Server Worker and is case sensitive.

Select the desired dependency option.
Select **Yes** if the worker is dependent on a system or previously registered custom worker. Select **No** if the worker is not dependent on another worker.

Click the Add Worker button/hot-spot.
If the **No** was select as the dependency option was specified, the worker is registered. It **Yes** was selected as the dependency option the list of registered workers is displayed. If you do not wish to add the worker click the **Discard Changes** button/hot-spot.

Place a checkmark in the checkbox adjacent to each worker on which the worker depends.
A dependency will be created for each worker selected. The Tx-Sync process will ensure that each worker selected completes processing before the new custom server worker is started.

Click the Create Dependency button/hot-spot to complete the registration.
If you do not wish to complete the registration, click the **Discard Changes** button/hot-spot.

To remove a Server Worker

Open the **Server Workers** tool by clicking on **Server Workers** on the Tools menu.

Click the **Removal** icon adjacent to the worker to be removed.
A message box will appear prompting you to confirm the removal.

Click the **OK** button to confirm removal of the worker.
If you do not wish to remove the worker click the **Cancel** button.

CHAPTER 7

Statistics

The **Statistics** menu option allows you to view data collected by the server regarding the performance of the Tx-sync process. The data collected can be used to gauge how well the system is performing at times of peak usage and also to identify areas of the synchronization process that might be degrading the performance of the overall system. Care should be taken when activating and monitoring statistics capture. The data collected by the server creates an additional load on the system and the tasks involved in statistics capture may themselves degrade the performance of the system.

To view Syncrhonization Statistics:

Select the **Statistics** option from the **Console Menu**.

Enter an integer value for the **Number of samples**.
The **Number of samples** value specifies the maximum number of syncrhonizations that will be included in the statics report.

Enter an integer value for the **Stats compute interval**.
The **Stats compute interval** indicates the length of time that the server will pause before recalculating all of the values included on the statistics report. A short interval causes the server to perform more frequent computations and may cause a degradation in performance if the server is heavily used at the time the statistics are calculated.

Click the **Start** button to display the statistics.
When the **Start** button is pressed the statistics report values are displayed. When the **Refresh** button is pressed the browser window is updated and the report values may change if more syncrhonizations have been processed and the server has recalculated the values.

Click the **Stop** button to end computation of statistics.
When the **Stop** button is pressed the statistics report is closed and the values are discarded.

Statistics Report Values:

Number of samples	the number of syncrhonizations used to calculate the disp

`Active`	the number of syncrhonizations being processed
`Queued`	the number of pending synchronizations; a pending syncl
`Sync Estimate`	the time in milliseconds being returned to devices indicat time that a syncrhonization may take to complete
`Sync Durations`	the time required to complete server processing of a sync
`Queued Durations`	the time spent by a syncrhonization in the server processi
`Worker Durations`	the time required to complete processing of the named se

CHAPTER 8

Server

The **Server** option allows you to view the current status of the server. The values displayed give an indication of the current load on the server and how server resources are being used. The following information is displayed when the Server option is selected from the Console menu:

`Memory Status`	displays memory usage on server: **Total Memory** - the total amount of memory assigned to the Java Virtual Machine **Used Memory** - the total amount of memory allocated by active server classes **Free Memory** - the total amount of memory availble for new allocations
`Pooled Object Status`	displays the usage levels of objects in various pools managed on the server: **Pooled Object / Class Name** - the fully-qualified class name of the object shared in the pool **In Use** - the number of objects currently referenced by a active server classes **Available** - the number of objects in the pool not assigne to an active class **Total** - the total number of objects in the pool
`Refresh`	when clicked the status information displayed on the screen is updated
`Force GC`	when clicked an attempt is made to run the Garbage Collector of the Java Virutual Machine; should result in reduction in *Used Memory*

CHAPTER 9

Tools

The **Tools** option provides access to a number of applications that are useful for deploying applications and custom objects on the system. The Tools menu also allows you to link your own custom applications to the Console by including them on this menu. The options available with the starndard installation are:

`Application Deployment`	tool for creating application archives that can be used to register applications on the installation; see Application Deployment
`Datapiping`	allows customization of the server's data piping process; see Data Piping
`Import / Export Utility`	allows server settings and applications to be saved to a file and retrieved from a file; see
`Obtain GUID`	tool for generating global unique identifiers; see Obtain GUID
`Server Workers`	registration tool for custom server workers; see Server Workers (on page 99)
`Soap Router Administration`	allows management of the servers internal SOAP router; see SOAP Router Administration

To Add a Link to the Tools Menu

Open the **Tools Menu** by selecting **Tools** from the Console Menu.

In the **Add New Link** seciton enter name of the application in the *Link* input box.
This text will appear in the Tools Menu such that when it is click it will start the application.

Enter the location where the application can be started in the URL input box.
This location must be in the format of a Uniform

Resource Locator (Eg. *http://localhost/mycustomtool*)

Click the **Add** button/hot-spot to add the new link to the menu.

To Remove a Link from the Tools Menu

Open the **Tools Menu** by selecting **Tools** from the Console Menu.

Click the **Removal** icon adjacent to the link to be deleted.
A message box will appear prompting you to confirm the deletion.

Click the **OK** button to remove the link from the menu.
If you do not wish to remove the link, click the **Cancel** button.

In This Chapter

Data Piping

The **Data Piping** tool allows you to create and manage the various components that make up the data piping process. The Data Piping tool allows you to define the structure of application data stored on central database systems and how this data will be represented on databases residing on the client device. Use of the Data Piping tool generally requires a solid understanding of the client applications being used in a solution since the client applications will normally access the databases and tables created by the data piping process. The following options are available on the **Data Piping Tool Menu**:

`Package Management`	tool for creating, editing and deleting data piping packages; see Managing Data Piping Packages
`Database Sources`	allows central database sources from which data is retrieved to be defined; Creating A Database Source
`Filtering Components`	tool for registering data piping filters to be used in conjunction with data piping packages; see Managing Filtering Components
`Client Databases`	allows creation of database configurations for client devices that are utilized by data piping packages; see Creating A Client Database

Creating A Database Source

The **Database Sources** option allows you to manage and configure the connection properties of the databases that are used by the data piping process. A Database Source is a set of JDBC connection parameters that allows the server to connect to a database during the datapiping process. Each data piping package is associated with a Database Source that identifies the location of tables referenced by the package. The **Database Sources List** displays the following information for each Database Source registered on the installation:

Name	text used to refer to the database source
Driver	JDBC driver used by the database source
URL	JDBC uniform resource locator used to identify database
User	database user used to connect to database

To create a Database Source

Click the **Add** button/hot-spot on the **Database Sources List** screen.
The Database Source registration from will appear.

If desired, select a database source from the drop down list to be used as a template.
If a database source is selected all of the remaining values on the form (except the *Password*) will be defaulted to the values used in the selected database source.

Enter the connection properties for the Database Source as follows:
All fields are required.

Database Source Name	text that is used to refer to the Database Source in the Console
Driver	the fully qualified name of the JDBC driver class to be used for this Database Source; Eg. *sun.jdbc.odbc.JdbcOdbcDriver* This class and any supporting classes must be included on the classpath used by the server.
URL	the JDBC uniform resource locator that identifies the database for the Database Source; Eg. *jdbc:odbc:atomadb*

`User`	the database user used to authenticate the connection
`Password`	the password for the user entered above

Click the **Accept** button/hot-spot to save the new database source.
If you do not wish to save the new database source click the **Discard** button/hot-spot.

Creating A Client Database

The **Client Database** option allows you to specify database creation parameters used by the data piping process to create databases on client devices. Each data piping package is associated with a Client Database. The table that is created from the data piping package will be created on the device in a database using the parameters specified by the Client Database. The **Client Database List** screen lists the following information for each database:

Name	text used to identify the Client Database; This text will appear wherever the Client Database is listed in the Console and will be used by client applications to open the database on the device.
DBMS	the type of database created on the device
Encrypted	indicates whether database is encrypted

To create a Client Database

Click the **Add** button/hot-spot on the **Client Database List** screen.
The Client Database registration form will appear.

If desired, select a Client Database from the drop down list to be used as a template.
If a Client Database is selected all of the remaining values on the form will be defaulted to the values previously used to register the selected Client Database.

Fill the fields of the form as follows:

Client Database Name	text used to identify the Client Database; This text will appear wherever the Client Database is listed in the Console and will be used by client applications to open the database on the device.

`DBMS`	the type of database that will be created on the device: **CDB** - native database on Windows Powered devices **SQL CE** - SQL Server for CE database; requires that SQL Server for CE be installed on the device
`Encrypted`	indicates whether database will be encrypted; only available on **SQL CE** databases; **Yes** - database is encrypted **No** - database is not encrypted
`Encryption Key`	seed or key string used to encrypt and decrypt data on an encrypted database

Click the **Accept** button/hot-spot to save the new client database.
If you do not wish to save the new client database click the **Discard** button/hot-spot.

Managing Data Piping Packages

A **Data Piping Package** is used to determine how a client device database table is structured and populated by the data piping process. The Data Piping Package is the core unit of the data piping process and contains all of the information necessary to retrieve data from a database source and to create the desired dataset for a client device based on the information retrieved.

Each Data Piping Package that is registered on an installation can be associated with an application. Once this association is made, the database table defined by the Data Piping Package is created and managed on any device where the application is deployed. When the last application that is associated with a Data Piping Package is uninstalled from the device, the table defined by the Data Piping Package is deleted from the device.

The **Package Management** option allows you to view the Data Piping Packages that have been registered on the installation and to edit and delete these registrations. The **Packages List** displays all of the registered data piping packages when the Package Management option is selected in the Data Piping tool. The following information is listed for each package:

`Name`	text used to identify the data piping package; when clicked the **Package Edit** form is displayed; see Editing and Cloning Packages & Deleting a Package
`Datasource`	name of the Database Source that defines the database used by the package
`Database`	the name of the Client Database where the table defined by the package will be created on client devices
`Table`	the name of the database table defined by the package
`Filtering Component`	the name of the Filtering Component (if any) associated with the package
`XML`	the XML source that defines the data piping package

Creating a Package

The database source to be used by a data piping package must be operational before a Data Piping Package is created. During creation of the package the server will connect to the database source to retrieve the database metadata describing the structure of the database objects defined on the source database.

To Create a Data Piping Package

Click the **Package Management** option on the **Data Piping** tool menu.
The **Package List** screen will appear.

Click the **Add** button/hot-spot on the **Package List** screen.
The Data Piping Package registration screen will appear.

Enter a name for the Data Piping Package.
The text entered here will be used to identify the Data Piping Package. This name must be unique as two packages on the same installtion cannot have the same name. This name will appear in any listing of all of the packages registered on the installation.

Set the maximum number of records in a Record Block generated by this package.
When the data piping process executes a package a dataset is generated including records of data that must be added to the device database table specified by the package. To facilitate processing of large amounts of data on the device, the records of the dataset are processed in blocks on the device. This value specifies the maximum number of records that will be included in a block.

(Optional) Select a Filtering Component to associate with the Data Piping Package if desired. The data piping filter selected will be executed each time the package is executed by the system. (see Managing Filtering Components)

Select the Client Database configuration to be used by the package.
The Client Database configuration determines where and how the table defined by the package will be created on the device. (see Creating A Client Database)

Enter the name of the client database table to be created by the package.
This entry determines the name of the table created

on the device. Client Applications will use this name in order to access the table.

Click the **Next** button/hot-spot to continue with the package definition.
If you do not wish to continue click the **Discard** button/hot-spot.

Select a Database Source from the list of sources registered on the installation.
The Database Source specifies the connection parameters that are used to retrieve data used in the package. Once a Database Source is selected the connection parameters being used are displayed. The information displayed cannot be edited.

Driver	fully qualified name of JDBC driver class
URL	the JDBC uniform resource locator indicating where the database is located
User	the database user used to authenticate the connection

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Select the tables that contain the source data to be included in the package.
All tables and views found in the source database are listed on the left. At least one source table must be included in the data piping package.
To add a table to the list of included tables, select the name of the table from the list on the left (name will be highlighted when selected) and click the > button.
To include all of the tables listed, click the >> button.
To remove a table from the list of included tables, select the table name in the Selected Tables list on the right and click the < button.
To remove all tables from the list of included tables click the << button.

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Select the columns from the source tables that will be included in the client database table defined by the package.
The columns defined in the tables previously included are listed on the left. Some columns may not be listed if the datatype used in the column is not supported by the system. The datatypes supported are **Integer, Memo, Float** and **VarChar**. A minimum of one column must be included in the client database table.
To add a column to the client database table, select the name of the column from the list on the left (name will be highlighted when selected) and click the > button.
To include all of the columns listed, click the >> button.
To remove a column from the client database table, select the name of the column in the Selected Fields list on the right and click the < button.
To remove all columns from the client database table click the << button.

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Create column relationships between the source tables if necessary.
When more than one source table has been included in the package, relationships between the columns of the tables must be defined. A relationship is equivalent to a Structured Query Language (SQL) "JOIN" definition between two tables. The relationship specifies how the records in one table are to be associated with the records in a second table. This is accomplished by selecting a column from the first table in the **Left Element** drop down list and selecting a column from a second table in the **Right Element** drop down list.
To create a relationship between the two columns click the > button.
To remove a relationship from the list of included Relations, click the < button.
To remove all relationships from the list of included Relations, click the << button.
Any record created in the client database table must satisfy all of the relationships defined here. The values in the columns of each source table must be equal in order to satisfy the relationship.

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Create the primary key of the client database table.
Select the columns that are to make up the primary key on the client device table. Any of the columns comprising the primary keys of the source tables, listed on the left, can be used. Columns included in the table may not be listed if the system does not support creation of primary keys on the column. A primary key column cannot allow **null** values and must be of type **Integer, Float** or **VarChar**. At least one column must be included as part of the primary key.
To add a column to the primary key, select the name of the column from the list on the left (name will be highlighted when selected) and click the > button.
To include all of the columns listed, click the >> button.
To remove a column from the primary key, select the name of the column in the Selected Fields list on the right and click the < button.
To remove all columns from the primary key click the << button.

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Enter aliases used as the names of columns in the client database table as desired.
The source columns are listed along with the aliases defining the names of the corresponding columns in the client database table. The aliases will be used by Client Applications that access the client database table and refer to the columns by name.

Click the **Next** button/hot-spot to continue with the package definition.
If you wish to return to the previous screen click the **Back** button/hot-spot. If you do not wish to continue click the **Discard** button/hot-spot.

Create indexes for the client database table as desired.

Any fields that may be indexed on the client database table are listed on the left. If a column was selected as part of a primary key for the table and index is automatically generated on the column. To add a column to an index, select the name of the column from the list on the left (name will be highlighted when selected) and click the > button. To include all of the columns listed, click the >> button.
To remove a column from an index, select the name of the column in the Selected Fields list on the right and click the < button.
To remove all columns from an index click the << button.
Enter a name for the index. This process can be repeated to create multiple indexes.

If you wish to view the document containing the package definition click the **XML Document** button/hot-spot.

Click the **Accept** button/hot-spot to register the data piping package.
If you wish to return to the previous screen click the **Back** button/hot-spot.If you do not wish to register the data piping package click the **Discard** button/hot-spot.

Editing and Cloning Packages

Editing a Data Piping Package

Through the **Package Management** option of the Data Piping tool any data piping package that is not used by an application can be edited. When a data piping package is assigned to an application using the Application Deployment tool, the package becomes locked from editing.

To Edit a Data Piping Package

Click the **Package Management** option on the **Data Piping** tool menu.
The **Package List** screen will appear.

Click the name of the package in the **Package List** screen.
The Data Piping Package edit screen will appear.

Follow the steps outlined in Creating a Package (on page 114) to edit the package.

Data Piping Package Clones

Any data piping package registered on the installation can be cloned using the Data Piping tool. When a data piping package is cloned, a new data piping package is created using the same source tables, source columns and client database table structure defined in the original data piping package. However the filter used by the package can be changed along with the names of the client database and client database table.

To Clone a Data Piping Package

Click the **Package Management** option on the **Data Piping** tool menu.
The **Package List** screen will appear.

Click the name of the package in the **Package List** screen.
The Data Piping Package edit screen will appear.

Click the **Clone** button/hot-spot.
The Package Clone creation screen will appear.

Enter a name for the cloned package.
Data Piping Package names must be unique. The name of the package being cloned is displayed and must be changed.

(Optional) Select a Filtering Component to associate with the Data Piping Package if desired.
The data piping filter selected will be executed

each time the package is executed by the system. (see Managing Filtering Components)

Select the Client Database configuration to be used by the package.
The Client Database configuration determines where and how the table defined by the package will be created on the device. (see Creating A Client Database)

Enter the name of the client database table to be created by the package.
This entry determines the name of the table created on the device. Client Applications will use this name in order to access the table.

Click the **Accept** button/hot-spot to register the cloned data piping package.
If you do not wish to register the data piping package click the **Discard** button/hot-spot.

To Delete a Data Piping Package

Click the **Package Management** option on the **Data Piping** tool menu.
The **Package List** screen will appear.

Click the name of the package in the **Package List** screen.
The Data Piping Package edit screen will appear.

Click the Delete button/hot-spot to remove the package from the installation.
A message box will appear prompting you to confirm the deletion of the package.

Click the OK button to delete the data piping package from the installation.
If you do not wish to delete the package click the Cancel button.

Deleting a Package

When a data piping package is assigned to an application using the Application Deployment tool, the package becomes locked from deletion. Before a Data Piping Package can be deleted, any application that uses the data piping package must be deleted (see Removing an Application). This ensures that the client database tables defined by the package are removed from the devices before the package is completely removed from the system.

To Delete a Data Piping Package

Click the **Package Management** option on the **Data Piping** tool menu.
The **Package List** screen will appear.

Click the name of the package in the **Package List** screen.
The Data Piping Package edit screen will appear.

Click the Delete button/hot-spot to remove the package from the installation.
A message box will appear prompting you to confirm the deletion of the package.

Click the OK button to delete the data piping package from the installation.
If you do not wish to delete the package click the Cancel button.

Managing Filtering Components

The **Filtering Components** option allows you to register data piping filters on your server. A Filtering Component is is an object that is used in conjunction with a data piping package to determine the data that is sent from a Database Source to a Client Database. Each data piping package can be associated with one Filtering Component. When the package is processed by the server, the Filtering Component is also called and is allowed to modify the dataset created from the package. The data piping process uses SOAP to call Filtering Components when necessary so each data piping filter must be registered in a SOAP router before it can by used by the system. The Filtering Components List displays the following information for each registered Filtering Component:

`Name`	the text used to refer to the data piping filter in the Console
`URL`	the location of the SOAP router where the data piping filter is deployed
`Object Name`	the object identifier under which the filter was deployed in the SOAP router
`Implementation`	indicates the SOAP convention that should be used to call the object
`Description`	a brief message associated with the filter

To register a Filtering Component

Click the Add button/hot-spot on the **Filtering Components List** screen.
The Filtering Components registration screen will appear.

If desired, select a Filtering Component from the drop down list to be used as a template.
If a Filtering Component is selected all of the remaining values on the form will be defaulted to the values previously used to register the selected Filtering Component.

Enter the properties of the Filtering Component as follows:

Filtering Component Name	the text used to refer to the data piping filter in the Console; this text will appear in any list of Filtering Components to identify this registration
URL	the location of the SOAP router where the data piping filter is deployed; Eg. *http://localhost:8080/rpcrouter*
Object Name	the object identifier under which the filter was deployed in the SOAP router
Implementation	indicates the SOAP convention that should be used to call the object **IBM** - use conventions normally associated with Java based routers **Microsoft** - use conventions normally associated with objects exposed throu Microsoft Internet Information Server
Description	a text message associated with the filtering component

Click the **Test Component** button/hot-spot if you wish to verify the information entered.
The server will attempt to locate the router and check that the object defined by the information entered exists.

Click the **Accept** button/hot-spot to register the new filtering component.
If you do not wish to register the new filtering component click the **Discard** button/hot-spot.

Editing a Filtering Component

Click on the name of the Filtering Component on the **Filtering Components List** screen.
The **Filtering Component Edit** form will appear.

Edit the values identifying the Filtering Component using the fields provided in the form.
A description of the avaliable fields appears above (see **To register a Filtering Component**)

Click the **Test Component** button/hot-spot if you wish to verify the information entered.
The server will attempt to locate the router and check that the object defined by the information entered exists.

Click the **Accept Changes** button to commit the Filtering Component changes.
If you do not wish to change the registration information, click the **Discard Changes** button.

To remove a Filtering Component

Click on the name of the Filtering Component on the **Filtering Components List** screen.
The **Filtering Component Edit** form will appear.

Click the **Delete** button/hot-spot below the fields in the edit form.
A message box will appear prompting you to confirm deletion of the Filtering Component registration.

Click the **OK** button to delete the Filtering Component registration.
If you do not wish to delete the registration, click the **Cancel** button.

Import / Export Utility

The **Import / Export Utility** can be used to transfer a variety of system settings from one installation to another. When the **Export** option is chosen, the settings of the current server are compiled into a ZIP archive that can be saved for a future import operation. When the Import option is chosen, the user is prompted for the location of a previously exported ZIP archive, then the utility reads the settings and data from the file and makes the apporpriate changes to the installations configuration. The Import / Export Utility archives and imports the following server information:

Data Piping	all of the settings configured using the Data Piping Tool are archived such as Filtering Component registrations, Data Piping Package registrations and Client Database configurations.
Settings	the parameters specified in the Client Settings and Server Settings options are archived such as synchronization timeout and retry values, synchronization error handling options and scheduler intervals
Applications	all application registrations are archived; see Applications (on page 73)
Groups	all Groups, their assigned Applications and associated data piping schedules are archived; see Groups
Custom Tool Links	all links added to the Tools menu are arhcived

To Export installation settings

Open the **Import / Export Utility** tool by clicking on **Import / Export Utility** on the Tools menu.

Click the **Export** button/hot-spot to begin the export process.
Your browsers *File Download* dialog box will appear.

Select the option to save the file from the *File Download* dialog.

Select the location where the file will be saved, rename the file if necessary and save the file.

To Import installation settings

Open the **Import / Export Utility** tool by clicking on **Import / Export Utility** on the Tools menu.

Click the **Browse** button below the *Import* section.
export process.
You will be prompted to navigate to the location of a previously exported archive file and to select the file.

After the previously exported archive file has been selected, click the *Import* button to import the archived server settings.

Obtain GUID

The Obtain GUID tool generates Global Unique Identifiers (GUIDs) for use in server extension objects. In order to create a server extension objects, such as Server Workers (on page 99), it is sometimes neccessary to implement a method in the object that returns a GUID. The GUID tool generates the GUID for you and displays it on the screen where it can be easily copied and included in your code where necessary.

To create a GUID

Open the **Objtain GUID** tool by clicking on it's name on the **Tools** menu.
A screen is displayed containing the GUID.

Copy the GUID and include it where needed.

SOAP Router Administration

The Simple Object Access Protocol (SOAP) allows applications to communicate using the HTTP protocol regardless of the language each individual application is written in or the software platform that each application is running on. The Atoma system makes extensive use of the SOAP protocol to facilitate communications between devices and Atoma servers and also between an Atoma server and acustom applications. This allows developers and system administrators to integrate a wide range of applications into an Atoma system.

Applications use a SOAP router as the destination for the the HTTP requests that carry their SOAP messages. The SOAP router listens for the these incoming messges and takes the required steps to service these requests. An Atoma server installation is equipped with a SOAP router to facilitate communications with the server. By default the router contains registrations of objects that are used internally by the Atoma system. The router allows adminsitrators to view these registrations and also add registrations of custom objects to the router.

The **View Services List** option is used to display the list of deployed web services and the details of each web service registration on the Atoma router.

The Adding a Service option is used to show the web service registration form that allows the deployment of a new web service.

Services List

The Services List lists all services that have been deployed on the server's router. The service *name* or registration *id* is shown in the list for each registered service. This list may include a number of registrations of system objects. These system registrations can be viewed but not deleted using the router administration tool.

To view details of a service

Click the on the name of the service that you wish to view.
The following information on the service registration is displayed:

ID	the service name
Scope	specifies how instances of the object are handled
Provider Class	full class name of Java provider
Use Static Class	inidcates whether exposed methods are static: **true** - methods are static (scope value is ignored if service uses static methods); **false** - methods are not static
Methods	listing of service's methods
Type Mappings	settings for how specific types are handled for service
Default Mapping Registry Class	class that will handle type conversions for the service

Click on the **View Services List** button/hot-spot to return to the services list.

To remove a service

Click the **Removal** icon adjacent to the service name in the Services List.
A messeage box will appear prompting you to confirm the service deletion.

Click **OK** to execute the deletion of the service.
If you do not wish to delete the service click the **Cancel** button.

Adding a Service

The **Add Service** option is allows registration of an object with the server's internal SOAP router. When an object is registered with the server's router, SOAP calls to the object can be sent to the server's Soap URL (see Client Settings).

To Add a Service to the internal router

If registering a Java object, ensure that the class and the classes used by the object are included on the classpath used by the server.
There are a number of ways of accomplishing this. One method is to add the jar(s) containing the desired classes to the "..\WEB-INF\lib" folder used by the Atoma web application folder. A restart of the server is usually necessary after performing this step.

Select the **Add Service** option after opening the **Soap Router Administraiton** tool on the Tools menu.
The **Service Deployment** form is displayed.

Fill in the required information on the **Service Deployment** form.
Some fields may be left blank depending on the type of service registered

ID	The name of the new service. Used as the *object name* in SOAP messages. must be unique among services currently registered on router
Scope	Determines how instancing of the object is handled: (i) **Request** - no sharing of object instances between requests to the router as each soap request will be executed on a new instance. (ii) **Session** - instance of object is shared for duration of the router's web session (iii) **Application** - instance of object is shared among all requests to the server
Methods	the name of each method in the object that is to be exposed by the service; method names are case sensitive
Provider Type	inidcates how the object was created: (i) **Java** - object is a Java class (ii) **Script** - object implemented via scripting code (iii) **User-Defined** - object exposed using a custom provider
Java Provider	parameters required when provider type is *Java*: **Provider Class** - full name of class (case sensitive); **Static** - **YES** if methods exposed are static **NO** if methods exposed are not static

Script Provider	parameters required when provider type is *Script*: **Script Language** - scripting language used in script **Script Filename** - name of file where script is saved **Script** - if script is not in a file the contents of the script are placed here
User-Defined Provider	parameters required when provider type is *User-Defined*: **Full Class Name** - full name of custom provider class **Number of Options** - total number of custom object options being specified; **Key/Value** - key and value pair of a custom option; options entered are passed directly to custom provider for application specific use
Type Mappings (optional)	allows custom handling of datatypes by the service **Encoding Style** - envelope encoding style used with type **Namespace** - xml namespace used to specify type in envelopes **Local Part** - name used to specify type in envelopes **Java Type** - corresponding Java language datatype **Serializer / Deserializer** - classes to handle the conversion of the type to and from XML envelopes respectively.
Default Mapping Registry Class (optional)	allows custom handling of datatypes; the class specified will be used to manage serialization and deserialization of datatypes to this service

Click the **Deploy Service** button/hot-spot to save the new service registration.
Click the **Clear Entries** button/hot-spot to remove all information from the form if you wish to re-enter all of the registration parameters. Click the **View Deployed Services** button/hot-spot to return to the **Services List** without registering the service.

CHAPTER 10

Tx-Sync

The **Tx-Sync Process** is the central event on which many of the systems features and services are built. The Tx-Sync process provides a reliable synchronization of system and application specific data between a client device and a server. Each syncrhonization performed between a client device and a server is treated as a single unique event. The Tx-Sync process operates on the principle that each syncrhonization event should be completed "successfully".

If a syncrhonization is aborted for any reason, the next attempt to synchronize by the client device will not establish a new synchronization but instead attempt to resume the previously aborted synchronization. The server resumes a previously aborted synchronization by trying to execute any server workers that failed when the syncrhonization was aborted or by continuing where the synchronization left off, if there was a communications error. When a synchronization is resumed the integrity of the data associated with the aborted syncrhonization is preserved. The device does not overwrite the previously transmitted data with a new set of data. For example, a device sends five application "transactions" to the server during a synchronization which is aborted. After this first sychronization aborts, the device user submits three additional application "transactions" and attempts to perform a syncrhonization. The syncrhonization that aborted is resumed and the last three application "transactions" remain in the pending synchronization queue on the device. The last three application "transactions" will be transmitted to the server during a second synchronization executed after the first aborted synchronization is successfully completed. The integrity of the data transmitted is guaranteed in the event that there is a dependency between the five application "transactions" originally transmitted and the three application "transactions" added after the synchronization aborted. Because of this built in functionality, it is generally a good idea to encourage device users to immediately perform a second synchronization after a resumed syncrhonization is completed successfully to ensure that all queued/pending data on the device is submitted to the server. A system administrator can also reset an aborted synchronization if desired (see Resetting a Device Sync). When the Tx-Sync process is reset for a device, a new synchronization is guaranteed to be created the next time that the device user initiates the Tx-Sync process. However, it should be noted that there is a potential for data loss when a synchronization is reset as the synchronization may have aborted before application data was committed or saved by the server.

The Tx-Sync Process can be customized to solve your specific application issues using a variety of extension objects such as the Synchronization Guard and Custom Server Workers and by taking advantage of system features such as Asynchronous SOAP Transactions and Data Piping Services .

In This Chapter

Asynchronous SOAP Transactions

An **Asynchronous SOAP Transaction** consists of a SOAP request that is sent from the client device during the Tx-Sync Process, is executed on behalf of the client device by the server, and the response is returned to the client device at the end on the Tx-Sync process. Asyncrhonous SOAP Transactions are processed by the **SoapWorker** during the execution of a Tx-Sync. Asynchronous SOAP Transactions allow client device applications to utilize virtually the same operating logic whether operating in a disconnected mode or connected mode. They also allow the Tx-Sync process to provide rich feedback to client applications regarding the transactions that have been submitted while in disconnected mode.

For example, an order entry system requires that five fields of information are reported and validated by a backend order processing system before becoming actual new orders. A server side object that communicates with the backend order processing system has been created and the object has a function called *CreateOrder* that takes the five fields of information as parameters. A client device application, developed to allow users to create new orders, calls the *CreateOrder* function after collecting the required information from the user.

This application can use the SOAP protocol to call the *CreateOrder* function when operating in connected mode using a synchronous SOAP call. The application can report the outcome of the attempt to create a new order using the return values returned during the SOAP call.

When in disconnected mode, this application can use the SOAP protocol to create an Asynchronous SOAP Transaction that will be executed on behalf of the application during the Tx-Sync process. When the synchronization is complete, this application can check on the outcome of the Asynchronous Soap Transaction and report the outcome of the attempt to create a new order using the return values returned during execution of the Asynchronous SOAP Transaction.

CHAPTER 11

Data Piping Services

Data Piping is the process of moving information from centralized database management systems to databases residing on the client device. Data Piping allows the information being sent to the client device database to be processed before it is sent to the device such that the structure of the data on the device can be controlled and the data can be filtered so that only the information necessary for a particular application or user is transfered to a device.

The Data Piping process does not take information stored in client database tables and attempt to reconcile this information with a corresponding central server database system. The process only takes information from central server database systems and transfers data to the target client devices during the Tx-Sync process. It is straightforward to see how Asynchronous SOAP Transactions can work in conjunction with the Data Piping process to provide highly intelligent data flow between a central server database and a client device. Asyncrhonous SOAP Transactions can be used to report data collected by a client device application to the central database, ensuring that the central database integrity is maintained and that the client device application receives feedback on the database updates. While the Data Piping process can be used to ensure that the latest source data from the central database is reflected on the databases stored on the client device.

Version Control & Monitoring

Version Control consists of the installation and removal of applications and system components from a client device. Each application registered on an installation is assigned a unique identifier. When an application is associated with a Group, an attempt will be made to install the application on any device synchronizing with the group. When the application is removed from the Group, an attempt will be made to remove the application from devices that have the application installed when the device synchronizes with the Group. To deploy a new version of an application, the files comprising the application must be registered on the installation using the Application Deployment tool. In effect, this creates a new application that can be added to the desired Group(s). Any previous version of the application can simply be removed from the appropriate Group(s) to ensure that it is no longer installed on any devices.

Monitoring consists of the information that is reported by the Device Framework to the server during the Tx-Sync process. One of the first items reported to the server during the synchonization process is the list of applications currently installed on the device and the operating condition of the device. After the Version Control process is completed, the list of applications is used to determine what is installed on the device and this can be viewed through the Inventory option. The operation condition of each synchronized device can be viewed using the Status option. The values of the parameters reported in the Status information reflect the condition of the device at the time of synchronization.

Custom Server Workers

A **Server Worker** is an object that performs tasks during the Tx-Sync process. The basic tasks that occur during a Tx-Sync such as application version control and execution of asynchronous SOAP transactions are all executed by Server Workers. These standard Server Workers are a part of the system and are installed and configured automatically. In addition to these standard Server Workers, custom Server Workers can be defined and registered on an installation. These custom Server Workers are Java classes that implement the Server Worker interface and are registered on the installation using the Server Workers (on page 99) tool.

Custom Server Workers can be used to perform virtually any server-side task desired during the synchronization process. One common use of custom server workers is the processing of application specific files that are sent from the device to the server during the synchronization. For example, in an application designed to capture customer signatures as images on the client device, the signature images are transferred to the server during the Tx-Sync process and a custom server worker is utilized to insert the captured images into a central database holding completed order information.

Synchronization Guard

The **Synchronization Guard** is a custom object that can be used to perform application or site specific tasks before the Tx-Sync process is started for a client device. The Synchronization Guard can be thought of as a custom security agent that authorizes or rejects a syncrhonization request from a client device. The Synchronization Guard does is global to the installation. The designated syncrhonization guard will be used to validate all syncrhonization requests directed at the server. This object is called using the SOAP protocol and as a result can reside on any server that will accept SOAP requests from the system server.

When a Synchronization Guard is registered on the server, through the Server Settings option, each synchronization request from a client device is first authorized by the Syncrhonization Guard. The TxSync Parameters (on page 153) entered by the user are passed to the guard objec and can be used to perform any custom validations desired before the synchronization begins. The guard object returns a value to the server indicating whether the synchronization is authorized or rejected. If authorized, the server continues with the normal synchronization tasks. If rejected the Tx-Sync process is stopped and the device user is notified.

Error Handling

The Tx-Sync process provides a number of **Error Handling** options that allow you to specify how the server responds to error conditions. As server workers perform the tasks included in the Tx-Sync process it is possible that program exceptions may occur. If a server worker throws an exception the synchronization will be aborted. Therefore as exceptions occur, server workers determine how to handle the exceptions based on the current Error Handling settings. The Error Handling settings are configured using the Server Settings option.

A system administrator can also reset an aborted synchronization if desired (see Resetting a Device Sync). When a syncrhonization is reset for a device, the Tx-Sync process is essentially re-initialized and a condition that previously caused the synchronization to abort might be eliminated. However, it should be noted that there is a potential for data loss when a synchronization is reset as the aborted synchronization may have stopped before application data was committed or saved by the server.

Tx-Sync Agent and Client Workers

The **Tx-Sync Agent** is the application that runs on the client device during the Tx-Sync process. The Tx-Sync Agent is responsible for collection of information from the user and the Device Framework components, transmission and retrieval of data from the server and performing any required tasks after data has been received from the server. The Tx-Sync Agent guides the device user though the synchonization process and provides a status of the syncrhonization as it is completed.

A **Client Worker** is an custom application that is executed on the client device during the Tx-Sync process. There are two types of Client workers, *pre-workers* and *post-workers*. A *pre-worker* is a client worker that is executed prior to transmission of data from the device to the server. One typical use of a *pre-worker* is to place application files in the device's outbox folder so that files will be transmitted to the server by the Tx-Sync Agent. A *post-worker* is a client worker that is executed after data has been received from the server during a synchronization. One typical use of a *post-worker* is to interpret files in the device inbox that have been sent from the server to the device. Client workers can be used to perform essentially any task that is desired on the client device during synchonization.

TxSync Parameters

TxSync Parameters are customizable values that are entered by a client device user prior to executing a synchronization. The values entered by the user are transfered to the server at the start of the Tx-Sync process. These values are passed to all custom synchronization objects such as the Synchronization Guard , Custom Server Workers and Data Piping Filters. These synchronization objects may use the parameters to customize the operation and results of the Tx-Sync process.

For example, in a delivery application a Tx-Sync parameter named *Route ID* is created. A data piping filter for the delivery application uses the *Route ID* value entered by the user to determine the set of delivery documents that is sent to the database on the user's device. Users who enter different *Route ID* values receive different delivery documents based on the value entered.

Tx-Sync Parameters are associated with Groups in the Console. Each group has its own set of parameters or prompts. If you would like the same parameter to appear in more than one group, it must be created separately in each group where desired.

The Tx-Sync Agent maintains a cache of the parameter values entered during the last synchronization. When a device user starts the Tx-Sync Agent, the Group with which the device last synchronized will be selected by default and the prompts for this group will be displayed. The values entered during the last synchronization will be filled by default at each prompt with the exception of prompts of type **password**. If a Tx-Sync parameter is designated as type **password** when it is created it will not be saved in the cache maintained by the Tx-Sync Agent.

CHAPTER 12

System Security

SSL

All system communications between a client device and an Atoma server are conducted using the HTTP protocol and as a result can be secured using Secure Sockets Layer (SSL). Depending on client device and server capabilities, SSL can offer up to 128-Bit encryption of the data transmitted between device and server.

To enable SSL communications on the system two tasks must be performed:

A server certificate must be installed on the web server and web context used by the Atoma installation.

The URLs configured in the Console for client communications must be formatted with the SSL (i.e. https) syntax.

Before the certificate can be installed it must be purchased from a trusted authority in the Public Key Infrastructure. Companies such as Verisign, Certicom or Thawte are some of the more popular authorities. The URL domain associated with the server certificate should match the URL domain to be used by client devices when communicating to the server installation.

The steps required to install a server certificate vary depending on the web server in use by the Atoma installation. Once installed, the URLs used by client devices to connect should be modified in the Console to reflect the use of SSL. These URLs are configured in the Console under Client Settings and Load Balancing . For example, the default Soap-URL specified in the Client Settings screen might be:

```
http://mycomputer:8080/atoma/SyncSvr
```

When secured with SSL this URL would be changed to:

```
https://mycomputer:8080/atoma/SyncSvr
```

Once these steps have been completed all system communications between the client device and server will be encrypted using SSL. The actual level of encryption is limited by the client device's capability however most devices supported by the system (WindowsÔ laptops, Windows CE devices) are capable of 128-Bit encryption.

Client Authentication

In addition to SSL, additional system security can be acheived through the use of Atoma Client Authenciation. Atoma Client Authentication is used to prevent unauthorized communication with an Atoma server from unknown clients. Atoma Client Authenciation is enabled in the Console on the Security screen.

An Atoma client certificate is deployed to any device that completes the setup process. The certificate deployed to the device adheres to the X.509 standard. When client authentication is enabled, any clients attempting to communicate to the server via the system URLs must possess a client certificate generated by the installation during a previously completed setup process. If the client does not possess a valid certificate, the communication request is rejected.

When Atoma Client Authentiction is enabled, the TxSync process can only be completed by devices that possess a valid client certificate.

CHAPTER 13

Copyright Notice

Copyright © 2001 Abaco.
All rights reserved

The software contains proprietary information of Abaco.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Due to continued product development this information may change without notice. The information and intellectual property contained herein is confidential between Abaco. and the client and remains the exclusive property of Abaco. If you find any problems in the documentation, please report them to us in writing. Abaco does not warrant that this document is error-free.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Abaco.

Microsoft Visual Basic, Microsoft Embedded Visual Basic, Windows®, Windows CE™,Windows 95™, Windows 98™, Windows NT® and Windows 2000™ are trademarks of the Microsoft Corporation.

The Apache Software License, Version 1.1

Copyright © 1999 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowlegement: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowlegement may appear in the software itself, if and wherever such third-party acknowlegements normally appear.

4. The names "The Jakarta Project", "Tomcat", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written * permission of the Apache Group.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Glossary of Terms

a

asynchronous soap envelope

application specific soap transaction that is executed during the Tx-Sync process ; client applications may use the Device Framework to queue soap transactions that are executed during the Tx-Sync process; in turn, client applications may then use the Device Framework to retrieve the responses from the queued soap transactions

auto-configuration

custom configuration program that may be deployed onto a media such as a compact flash card to allow configuration settings to be a client device simply by inserting and/or connecting the media to the client device

c

classpath

JavaÔ virtual machine variable that specifies the location of JavaÔ classes such that they may be loaded by the virtual machine when referenced

client worker

client application that is executed on the device during the Tx-Sync process ; there are two types of client workers: pre-workers are executed before information and files are transfered to an Atoma server; post-wokers are executed after information and files have been received from an Atoma server

connected application

a client application designed to function while a device has access to a network; connected applications validate most user entries against a remote server and post any information saved or submitted by the user to a remote server

d

data piping

process used by Atoma server to deploy and maintain databes on a client device; desired data is extracted from any number of centralized database servers and then formatted as desired and delivered to a device; after initial creation on device only databases changes (delta) are transferred to the device with each synchronization

data piping filter

custom programming object that is executed during the data piping process; data filter is able to limit and qualify the database records that are transferred to the device

D

Data Piping Scheduler

component on a server that triggers preparation of data packages based on the data piping schedule; component checks the schedule at a specified interval and executes the preparation of any data packages scheduled at the time

d

device

client hardware such as a Data Collection Terminal, Personal Digital Assistant or laptop computer that connects to an Atoma server

device type

label that describes the hardware, operating system and possibly the manufacturer and model of a device; device types are divided into two categories: generic device types & specific device types; see also generic device type and specific device type

disconnected application

a client application designed to function while a device does not have a network connection; disconnected applications use a local database and/or the local file system on the device to validate user entries and to save data submitted by the user; disconnected applications typically rely on a synchronization process to handle transfer or data saved or submitted by the user to a remote server;

g

generic device type

category of device that does not identify the manufacturer and model of the device; see also device type

J

JDBC

Java Database Connectivity; an API for accessing relational and tabular data (primarily databases) from the JavaÔ programming language

JDBC URL

string that specifies the location of a JDBC data source

s

server worker

custom programming object that is executed as each Tx-Sync process occurs; worker is able to recieve files from the device, receive parameters enterd by user, transfer files to the device and perform any desired application-related task

setup procedure

process by which the Atoma Device Framewok is installed onto a device; requires no manual installation of files; instead browser is used to access an Atoma server and after user authentication, necessary files are downloaded and automatically installed on the device

S

SOAP

Simple Object Access Protocol; lightweight mechanism of information exchange between remote applications

SOAP router

server application that listens for incoming SOAP messages; SOAP router is responsible for executing or forwarding SOAP messages as needed and returning required responses to the requestor.

s

specific device type

category of device that identifies the manufacturer and model of the device; see also device type

synchronization parameter

value entered by device user before beginning the Tx-Sync process ; value is transfered to the Atoma server and can be read an utilized by a data filter or server worker .

T

Tx-Sync Agent

component in the Device Framework that executes and manages the Tx-Sync process from a client device; this component handles execution of sync related tasks on the device and communications with a server during the process

Tx-Sync process

transactional synchronization process occuring between a client device and an Atoma server; application version control, device configuration, asynchronous SOAP execution, and data piping all occur during Tx-Sync; process can be extended using server workers

Index

Contents

CHAPTER 1

Introduction

Welcome

Thanks for you interest in developing with the Atomaä mobile system!

In this guide you will find documentation on the application programming tools and features provided for developers and guidelines for accomplishing the most common tasks executed in the system. The overall goal of the programming tools described herein is to allow developers to create rich mobile applications that can communicate with backend systems efficiently, securely, and with any desired frequency and communications mechanism. In support of this goal the system provides tools on both the server and the device for application development and data management.

The tools provided for device (client) development are designed to support development in mulitple programming languages and provide a simplified programming experience accross different device platforms and device manufacturers. These tools consist primarily of components that address the most common issues facing mobile application developers and provide features that allow applications to take advantage of the services provided by the overall system.

On the server, the system provides several features that enable easy integration of application objects with the system provided services. By following a few simple guidelines your server based application components can be "mobile-enabled" while you as a developer enjoy unparralleled freedom of choice among the programming languages and platforms available today and even those coming in the future.

In This Chapter

SECTION 1

Getting Started

A Developer's Kit is provided for each supported device platform. These developer's kits allow you to write applications that will reside on the device. The developer's kits do not include any development environments as it is generally your choice as to which language and development tool you will use to write your application.

While it is not necessary to have a server installation available during device application development, access to a server may be useful in the latter stages of the development process. If you will be installing a test server as part of your development environment, consult the server documentation for instructions on performing this installation.

Requirements

Each developers kit requires approximately 10 Megabytes of free hard disk space. The platform specific requirements and recommendations are listed below.

Windows Powered Devices

The following items are recommended for Windows Powered device development:

- Microsoft Embedded Visual Tools 3.0
- Microsoft Active Sync 3.1 or above
- Microsoft Platform SDK for Pocket PC

Developer's Kit Installation

When installing a Developer's Kit be sure to pick the kit designed for your target installation machine.

Windows Powered Devices

The kit supports development in Microsoftä Embedded Visual Basic, Microsoftä Embedded Visual C++, and using Application Server Pages technology. The kit installs desktop versions of the device framework components to allow early-binding and Microsoft IntelliSenseä in the Embedded Visual Basic development environment. Header files necessary for C++ development and CAB files for manual deployment of the framework to development devices are also installed.

Follow the instructions in the installation script to install the development kit. To manually install the framework on a development device, copy the appropriate cab file from the "sdk\wce\Cabs" subdirectory of the kit's installation folder to the target device and execute the file.

The development kit for Windows Powered devices also includes a set of eMbedded Visual Basic modules (".bas" files) that contain definitions of error constants/enumerations and functions which perform common tasks to simplify use of the framework from eVB. You may include these modules directly in your eVB project or selectively copy any desired definitions into your own code modules.

See Also

Additional Resources....

SECTION 2

Additional Resources

The Atoma product allows you to make use of a number of products and technologies from a variety of organizations. Below you will find a number of resources that provide helpful information related to the Atoma development experience.

Abaco Developer Support
"http://www.abacomobile.com/"

Microsoft Mobile Developer Center
"http://www.microsoft.com/mobile/developer"

Microsoft Visual Studio: Device Development
"http://msdn.microsoft.com/vstudio/device"

Microsoft Windows Embedded Developer Center
"http://msdn.microsoft.com/library/default.asp?url=
/nhp/Default.asp?contentid=28000437"

Microsoft Windows CE Operating System
"http://www.microsoft.com/windows/embedded/ce"

FAQs & Highlights for Embedded Visual Basic 3.0 and
Windows CE Toolkit for Visual Basic
"http://support.microsoft.com/support/default.asp?P
R=wincevb&FR=0&SD=MSDN&LN=EN-US&CT=SD&SE=NONA"

SOAP Specification "http://www.w3.org/TR/SOAP/"

See Also

Getting Started..............

CHAPTER 2

Pocket PC Device Framework

The Pocket PC Device Framework provides a set of components that simplify application development for a wide range of devices based on the Microsoftä Pocket PC platform. This set of components primarily consists of Component Object Model (COM) objects which are easily utilized by applications written in any of the development languages supported on Pocket PCs. These components will be installed automatically on any device that is configured for use in the system.

The Program ID of each object defined by a Device Framework library follows the description of the object Use the Program ID when creating instances of each object for use in your applications.

Supported Devices

- Standard Consumer Pocket PC 3.0

  Compaq iPAQ, Casio Cassiopeia, Hewlett Packard Jornada 540 etc... The Socket In-Hand Scan card is supported on all devices in this category.

- Intermec Technologies

  700 (Stingray), 70 series
- Symbol Technologies

  2800 Series, 8100 Series
- Casio Limited

  IT700 series

Using Microsoft Embedded Visual Basic

To include a device framework component in the Embedded Visual Basic IDE, use the `Projet|References...` dialog to add the component to your project. For example, to use the scanner object, add the abdevio component library to the project's references. Note that the name of the desktop version of the signature capture control is *abImgCapNT.dll* not *abImgCap.dll.*

Embedded Visual Basic supports late-binding through the *CreateObject* statement and so it is also possible to use the framework components without explicitly adding them to your project.

Using Microsoft Embedded Visual C++

The header and definition files for C++ development with device framework components are located in the "sdk\wce\include" subdirectory of the developer's kit installation. Add the appropriate .h and .c files to your project to access the COM intefaces of the desired components.

In This Chapter

CHAPTER 3

abAspex (ASP Extension)

The ASP Extension library provides an Application object that allows data to be saved between sessions of an application. A session is typically defined as the period during which a user works with an application i.e from the time the application is opened by the user until the time the application is closed. As the name implies, this object is particularly useful in ASP applications however the object may also be used from non-ASP applications.

The **Application** object provides a means of saving and retreiving values from any point within an application. The values stored by the Application object are saved between sessions of the application and after a warm-reboot of the device. The object provides a collection called Contents that is used to manage the values stored for the application.

The Program ID for this object is **abaspex.Application**.

Properties	Methods
Application::AppName	Application::Lock ....
	Application::Unlock ......

SECTION 3

Application::AppName

The **AppName** property is used to identify the set of values that will be saved and retrieved by the current instance of the Application object. Once a name has been applied to the AppName property, subsequent calls to the Application object's methods and properties will refer to the set of values identified by the specified name.

```
Visual Basic Syntax
```

objApplication.AppName = strName
See Also
Application::Lock.........
Application::Unlock .....

SECTION 4

Application::Lock

The **Lock** method turns on exclusive mode. In exclusive mode other processes are prevented from modifying the values associated with the current application. For example, this prevents two instances of the same application from attempting to change values simultaneously. The Application object stores values for all applications in a central datastore. It is possible, if desired, for two separate applications to modify the same values. This kind of open acces can be controlled by using the Lock method.

The AppName property must be set before the Lock method is called.

`Visual Basic Syntax`

objApplication.Lock
See Also
Application::AppNam
e ....
Application::Unlock .....

SECTION 5

Application::Unlock

The **Unlock** method turns off exclusive mode, allowing other processes to modify the values associated with the current application. When exclusive mode is off, any process that acquires a reference to the current application's values is able to modify the values. Typically calls to the Unlock method are preceded by a call to the Lock method .

The AppName property must be set before the Unlock method is called.

Visual Basic Syntax

objApplication.Unlock

See Also

Application::AppName ....

Application::Lock.........

The **Contents** property returns a collection of the values that are associated with the current application. This collection allows you to create, set, retrieve and remove application values.

Visual Basic Syntax

colContents = objApplication.Contents

Properties	Methods
Contents::Count............	Contents::Remove.........
Contents::Item..............	Contents::RemoveAll....

SECTION 6

Contents::Count

The **Count** property returns the total of the number of values in the Contents collection.

Visual Basic Syntax

```
lngCount = objApplication.Count
```

See Also

Contents::Item ....
Contents::Remove ....
Contents::RemoveAll....

SECTION 7

Contents::Item

The **Item** method is used to create, retrieve, or modify key-value pairs associated with the current application.

Each value stored by an application must be referenced with a key. The key is a string identifier used to refer to the value much like a variable name. When setting a value, if the key specified does not exist, a new key-value pair is created.

Setting a value

`Visual Basic Syntax`

objApplication(strKey) = val
objApplication.Contents.Item(strKey) = val

Retreiving a value

`Visual Basic Syntax`

val = objApplication(strKey)
val = objApplication.Contents.Item(strKey)

See Also

Contents::Count............
Contents::Remove ....
Contents::RemoveAll....

SECTION 8

Contents::Remove

The **Remove** method deletes an existing key-value pair.

```
Visual Basic Syntax
```
objApplications.Contents.Remove(strKey)

See Also

Contents::Count............
Contents::Item..............
Contents::RemoveAll....

SECTION 9

Contents::RemoveAll

The **RemoveAll** method removes all of the key-value pairs associated with the current application.

```
Visual Basic Syntax
```
objApplication.Contents.RemoveAll

See Also

Contents::Count............
Contents::Item..............
Contents::Remove........

CHAPTER 4

abSoapQueue (Web Service Queue)

The **absoapqueue** library provides tools for utilizing web services in applications designed for disconnected environments. Communication using the SOAP protocol is a key element of the device framework and is primarily achieved using the absoap (SOAP)) library. The absoap library operates in a synchonous manner thus an application that transmits a request using this library is forced to wait until a response to the request is received. The absoapqueue library enables asynchronous SOAP communications where the request and response envelopes are managed in an internal queue. By managing SOAP enevlopes in this manner, applications designed to operate without network connectivity are able to utilize the system's Tx-Sync process to communicate while maintaining the integrity of their requests.

In This Chapter

SoapQ

The **SoapQ** object allows applications to work with a queue of SOAP requests to be processed asynchronously. Requests (outgoing) and responses (incoming) are managed separately. Applications are able to associate a custom identifier with each request added to the queue that can later be used to retrieve the appropriate response after synchronization with a server has occurred. During synchronization, requests are executed in the order in which they are added to the queue allowing application(s) to maintain data and logic integritry even while operating asynchronously.

The Program ID for this object is **absoapqueue.SoapQ**.

Properties

SECTION 10

SentRequests

The **SentRequests** method returns a collection of all of the requests that have been submitted for processing during a synchronization.

`Visual Basic Syntax`

Set objRequests = objSoapQ.SentRequests

See Also

SECTION 11

QueueName

The **QueueName** property sets the queue to be managed by the current instance of the SoapQ object. This property is a string value used to uniquely identify an application and correspondingly the requests and responses that belong to the application. If the QueueName property is set to a value which does not match an existing application, a new application space is created for the application. If the property is set to a value of an existing application, the current instance of the SoapQ object will be set to manage the queue that exists for the specified application.

This property can also be used to retrieve the name of the queue currently in use.

`Visual Basic Syntax`

objSoapQ.QueueName = strName
See Also
SentRequests ....
Requests ....
Responses ....

SECTION 12

Requests

The **Requests** property returns a collection of all of the SOAP requests stored in a queue. A request is a SOAP envelope which is sent to a server to process a remote call.

`Visual Basic Syntax`

Set objRequests = objSoapQ.Requests

`Error Values`

ABERR_COLLECTIONINIT	Internal object has not been initialized.

See Also

SentRequests ....
QueueName..................
Responses.....................

Responses

The **Responses** property returns a collection of all SOAP responses stored in a queue. A response is a SOAP envelope which is returned to a client in response to a remote call.

Visual Basic Syntax

Set objResponses = objSoapQ.Responses

Error Values

ABERR_COLLECTIONINIT	Internal object has not been initialized.

See Also

SentRequests ....
QueueName..................
Requests ....

Request

The **Request** object allows applications to manage request items in the soap queue. A request can be added or retrieved from the queue using the Requests object.

The Program ID for this control is **absoapqueue.Request**.

Properties	Methods
Agent............................	SetPayload ....
UserDef........................	GetPayLoad ....
Encrypted	DeleteHeader ....
URL	
Name............................	
Header..........................	
DeleteAfterSync	

SECTION 14

Agent

The **Agent** property is reserved for future use.

```
Visual Basic Syntax
```
objRequest.Agent = strName

See Also

Name............................
URL ....
UserDef........................
Encrypted ....
GetPayLoad..................
Header..........................
DeleteAfterSync ....
DeleteHeader................
SetPayload....................

SECTION 15

Name

The **Name** property is used to retrieve the identifier assigned to the request by the queue This identifier is used to retrieve and/or delete the request from the queue. This name identifier is also assigned to the response corresponding to the request.

Visual Basic Syntax

strName = objRequest.Name

See Also

Agent............................
URL ....
UserDef........................
Encrypted ....
GetPayLoad..................
Header..........................
DeleteAfterSync ....
DeleteHeader................
SetPayload....................

Section 16

URL

The **URL** property holds the Uniform Resource Locator of the destination router for a request item. This URL specifies the network location where the target web service is published.

Visual Basic Syntax

```
objRequest.URL = strURL
strURL = objRequest.URL
```

See Also ·

Agent............................
Name............................
UserDef........................
Encrypted......................
GetPayLoad..................
Header..........................
DeleteAfterSync...........
DeleteHeader................
SetPayload....................

SECTION 17

UserDef

The **UserDef** property is used to set and retrieve a custom string associated with the request. This string is also assigned to the response received when the request is processed.

Visual Basic Syntax

```
objRequest.UserDef = strValue
strValue = objRequest.UserDef
```

See Also

Agent............................
Name............................
URL ....
Encrypted ....
GetPayLoad.................
Header..........................
DeleteAfterSync ....
DeleteHeader................
SetPayload....................

SECTION 18

Encrypted

The **Encrypted** property returns a flag indicating if the specified Request is encrypted before storage.

Visual Basic Syntax

```
boolValue = objRequest.Encrypted
```

See Also

Agent............................
Name............................
URL ....
UserDef.........................
GetPayLoad..................
Header...........................
DeleteAfterSync ....
DeleteHeader................
SetPayload....................

SECTION 19

GetPayLoad

The **GetPayLoad** property returns the XML string representation of the web service request represented by the Request object. This string can be loaded into the SoapProxy object and the methods and properties of the SoapProxy object can in turn be used to manipulate the item.

Visual Basic Syntax

```
strXML = objRequest.GetPayLoad
```

Error Values

ABERR_DETAILFAILED	Unable to retrieve item data.

See Also

Agent............................
Name............................
URL.............................
UserDef.........................
Encrypted......................
Header..........................
DeleteAfterSync...........
DeleteHeader................
SetPayload....................

SECTION 20

Header

The **Header** property allows an application to set and retrieve HTTP header values associated with the web service request. When the request is executed the HTTP protocol is used to communicate with the specified Web Service. The Header property allows any custom headers associated with the underlying HTTP request created when the web service request is executed.

Visual Basic Syntax

objRequest.Header strName, strValue
strValue = objRequest.Header strName

See Also

SECTION 21

DeleteAfterSync

The **DeleteAfterSync** property specifies how storage of the Request object will be handled by the client framework. This property is a boolean value indicating whether the Request object will be automatically deleted when it is submitted for processing during the TxSync process. A true value indicates that the framework will handle deletion of the object on behalf of the application when the request is completely processed. A false value indicates that the application will take responsibilty for deleting the object when it is no longer needed.

`Visual Basic Syntax`

objRequest.DeleteAfterSync boolValue
boolValue = objRequest.DeleteAfterSync

See Also

Agent............................
Name............................
URL ....
UserDef........................
Encrypted ....
GetPayLoad..................
Header..........................
DeleteHeader................
SetPayload...................

SECTION 22

DeleteHeader

The **DeleteHeader** method removes the specified HTTP Header value from the Headers collection associated with the Request.

`Visual Basic Syntax`

objRequest.DeleteHeader strName
See Also
Agent............................
Name............................
URL.............................
UserDef........................
Encrypted......................
GetPayLoad.................
Header..........................
DeleteAfterSync...........
SetPayload....................

SetPayload

The **SetPayload** method assigns an xml envelope representing a web service request to an instance of the Request object.

Visual Basic Syntax

objAsyncPostItem.SetPayload strXML

See Also

Agent............................
Name............................
URL ....
UserDef........................
Encrypted ....
GetPayLoad..................
Header..........................
DeleteAfterSync ....
DeleteHeader................

Response

The **Response** object allows applications to manage responses corresponding to request items stored in the soap queue. A response can be retrieved from the queue via the Responses object.

SECTION 24

Agent

The **Agent** property is reserved for future use.

Visual Basic Syntax

strName = objRequest.Agent
See Also
UserDef........................
GetPayload...................
Encrypted.....................

UserDef

The **UserDef** property returns the custom string associated with the request corresponding to a response. This string can be assigned to the request through the UserDef property of the Request object.

Visual Basic Syntax

strValue = objRequest.UserDef

GetPayload

The **GetPayLoad** property returns the XML string representation of the web service response represented by the Request object. This string can be loaded into the SoapProxy object and the methods and properties of the SoapProxy object can in turn be used to manipulate the item.

Visual Basic Syntax

strXML = objRequest.GetPayLoad

SECTION 25

Encrypted

The **Encrypted** property returns a flag indicating if the specified response is stored in an encrypted.

`Visual Basic Syntax`

boolValue = objRequest.Encrypted

See Also

Requests

The **Requests** object provides access to all web service requests stored in a queue. The Requests object allows applications to access individual requests and also remove requests from the queue.

Methods

Section 26

Add

The **Add** method inserts a new web service request into the queue. An instance of the Request object containing the web service request is passed as a parameter to the method.

```
Visual Basic Syntax
objRequests.add objRequest
```

See Also

Delete ....
DeleteAll ....
Exists............................

SECTION 27

Delete

The **Delete** method permanently removes an item from the queue. The Delete method takes the identifier assigned to the request as a parameter.

```
Visual Basic Syntax
```

objRequests.Delete strName

See Also

Add ....
DeleteAll ....
Exists............................

SECTION 28

DeleteAll

The **DeleteAll** method permanently removes all of the request items stored in the queue.

```
Visual Basic Syntax
```

Requests.DeleteAll

See Also

Add ....
Delete ....
Exists............................

SECTION 29

Exists

The **Exists** method returns a flag indicating if the specified identifier corresponds to a request item in the queue.

```
Visual Basic Syntax
```
boolValue = objRequests.Exists(strName)

See Also
Add ....
Delete ....
DeleteAll ....

Responses

The **Responses** object provides access to all web service responses stored in a queue. The Responses object allows applications to access individual responses and also remove responses from the queue.

SECTION 30

Delete

The **Delete** method permanently removes a response item from the queue. The Delete method takes the identifier assigned to the response as a parameter.

```
Visual Basic Syntax
```

objResponses.Delete strName

See Also

DeleteAll......................
Exists............................

SECTION 31

DeleteAll

The **DeleteAll** method permanently removes all of the response items stored in the queue.

```
Visual Basic Syntax
```

Responses.DeleteAll

See Also

Delete............................
Exists............................

SECTION 32

Exists

The **Exists** method returns a flag indicating if the specified identifier corresponds to a response item in the queue.

```
Visual Basic Syntax
```

boolValue = objResponses.Exists(strName)

See Also

Delete...........................

DeleteAll......................

CHAPTER 5

abDevio (Scanner, Magnetic Card Reader)

The **abdevio** library provides developer access to hardware peripherals whose underlying programming APIs differ from manufacturer to manufacturer. The hardware peripherals supported by the abdevio library are input/output peripherals such as scanners and magnetic stripe readers(MSR) which improve the ease of operability of an application and increase data i/o accuracy. The abdevio library allows applications written with the Device Framework to take advantage of device specific capabilites while maintaining portability accross devices built by a number of manufacturers.

The **Scanner** object allows applications to receive input from and configure barcode scanners. The Scanner object uses an event-driven programming model to notify applications when data has been read by a device scanner.

The data read through the scanner control is accessible through the OnDataAvailable event associated with the **Scanner** control itself. The actual text and symbology of the data read by the scanner can be retrieved by calling the GetData and GetSymbologyMask methods respectively. To set which barcode symbologies will be read by the scanner, use the SymbologyMask property.

The Program ID of the this object is **abdevio.Scanner**.

Properties	Methods	Events
Scanner::DataAvailable....................................	Scanner::GetData..........	Scanner::OnDataAvailable..............................
Scanner::Enabled..........	Scanner::GetSymbologyMask.........................	
Scanner::Supported.......	Scanner::SoftTrigger.....	
Scanner::SymbologyMask...............................		
Scanner::TriggerKey.....		
Scanner::WedgeMode...		

SECTION 33

Scanner::DataAvailable

The **DataAvailable** property returns a boolean value indicating whether information has been read by a scanner. If the value is true, data has been read by the scanner. If the value is false, no data has been read. If data has been read by the scanner, the data can be accessed by calling the GetData (Scanner) method.

Visual Basic Syntax

boolValue = objScanner.DataAvailable

See Also

Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::OnDataAvail
able...............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

SECTION 34

Scanner::Enabled

The **Enabled** property sets or retrieves the status of the Scanner object. If the value is set to true, Scanner is activated. Upon activation the scanner engine settings are applied and any native libraries required by the scanning API are loaded. If the value is set to false, the scanner libraries are unloaded and your application's Scanner object will no longer respond when input is received by the scanner hardware.

Visual Basic Syntax

objScanner.Enabled = boolValue

Error Values

ABERR_SCANENGINE	Error creating or loading scanner engine.
ABERR_HWDLL	Error loading scanner api.
ABERR_ENGINEINIT	Error initializing scanner engine.
ABERR_SETSYMBOLOGY	Error setting barcode symbologies.
ABERR_BARCODENONE	Symbology mask has not been set to som than devioBARCODE_NONE.

See Also

Scanner::DataAvailable....
Scanner::GetData..........
Scanner::GetSymbologyMask........................
Scanner::OnDataAvailable..............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyMask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

SECTION 35

Scanner::GetData

The **GetData** method returns the data held in the scanner's reading buffer. The scanner's reading buffer is filled with data each time a successful scan is completed. This method returns a string value equivalent to the data last read by the scanner. After this method is called, the data is removed from the Scanner object's reading buffer.

Visual Basic Syntax

```
strValue = objScanner.GetData
```

See Also

Scanner::DataAvailabl
e ....
Scanner::Enabled ....
Scanner::GetSymbolo
gyMask........................
Scanner::OnDataAvail
able..............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

SECTION 36

Scanner::GetSymbologyMask

The **GetSymbologyMask** method returns the barcode symbology flags associated with the last successful scan. This method returns a long value which can be compared with the values in the devioBARCODE enumeration to determine the symbology scanned. Most scanners return a value which indicates the single symbology that was scanned. Other scanners return a number of possible symbologies. Check the Scanner Notes further information regarding specific scanner models.

Visual Basic Syntax

```
lngValue = objScanner.GetSymbologyMask
```

See Also

Section 37

Scanner::OnDataAvailable

The **OnDataAvailable** event is fired when a scan is completed successfully on a device. This event is only fired when the scan is successful. If the hardware's attempt to decode the scan is not successful, the event will not fire since no data will be read. The procedure in your application that is associated with this event can be used to determine the symbology and to retrieve the data that was read using the GetData (Scanner) and GetSymbologyMask methods of the Scanner object.

Visual Basic Syntax

```
Set objScanner = CreateObjectWithEvents("abdevio.Scanner", "Scanner_")
...
Sub Scanner_OnDataAvailable()
...
End Sub
```

See Also

Scanner::DataAvailabl
e ....
Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes...............

SECTION 38

Scanner::SoftTrigger

The **SoftTrigger** method is used to execute a scanner read operation. When this method is called, the scanner harware will attempt to read data. Most devices equipped with integrated scanners have a trigger button that allows the user to initiate a scanner read. This method triggers the scanner in the same way that the hardware button would trigger the scanner.

This method provides one technique of triggering the scanner on devices that are not equipped with a trigger button. Another technique, is to associate an existing device button with the scanner using the TriggerKey property .

Visual Basic Syntax

```
objScanner.SoftTrigger
```

Error Values

PYLON_E_SCANENGINE_FAILED	Error creating or loading scanner engine.
PYLON_E_HWDLL_LOAD_FAILED	Error loading scanner api.
PYLON_E_ENGINE_INIT_FAILED	Error initializing scanner engine.
PYLON_E_SET_SYMBOLOGY_FAILED	Error setting barcode symbologies.
PYLON_E_ENGINE_ERROR	An unexpected engine ocurred.
PYLON_E_NO_BARCODE_SET	Symbology mask has not been set to som than devioBARCODE_NONE.

Remarks

This method attempts to enable the scanner if needed.

See Also

Scanner::DataAvailabl
e.................................
Scanner::Enabled.........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::OnDataAvail
able..............................
Scanner::Supported.......
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

SECTION 39

Scanner::Supported

The **Supported** property returns a value indicating whether the currently installed library contains a complete Scanner object implementation. Certain devices supported by the Device Framework cannot be equipped with hardware scanners and as a result the **abdevio** library that is automatically installed on the device is not capable of working with a hardware scanner. If this property returns true, the object contains a scanner implementation. If the property returns false, the object does not contain a scanner implementation and the Scanner object should not be used.

This property does not verify if the API libraries or the scan hardware is properly installed on the device.

Visual Basic Syntax

```
boolValue = objScanner.Supported
```

See Also

Scanner::DataAvailabl
e..................................
Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask........................
Scanner::OnDataAvail
able..............................
Scanner::SoftTrigger.....
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

Section 40

Scanner::SymbologyMask

The **SymbologyMask** property is a read/write property used to set and retrieve the barcode symbologies that are enabled on the hardware scanner. When a barcode symbology is enabled, the scanner hardware will decode and succesfully read a barcode printed in the specified symbology. When a symbology is disabled, the scanner is unable to successfully read the specified barcode.

The SymbologyMask property should be set before enabling the scanner with the Enabled (Scanner) property. Symbologies can be specified using the devioBARCODE enumeration which assigns a value to each supported type.

```
Visual Basic Syntax
```

```
objScanner.SymbologyMask = lngValue
```

To enable a group of symbologies combine the each of the appropriate values using an *OR* statement and assign the resulting value to the SymbologyMask property.

```
'Symbology constants (can be found in mabdevio.bas)
Const Code39 = 1
Const I2of5 = 8
Const UPC_A = 512

'declare scanner object
Dim oScan as ABDEVIO.Scanner

'create scanner object with event support
Set oScan = CreateObjectWithEvents("Abdevio.Scanner", "Scanner_")
'Set desired symbologies
oScan.SymbologyMask = Code39 Or I2of5 Or UPC_A
....
'Remove I2of5 and UPC
oScan.SymbologyMask = Code39
...
```

See Also

Scanner::DataAvailabl
e ....
Scanner::Enabled ....
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::OnDataAvail
able..............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::TriggerKey.....
Scanner::WedgeMode...
Scanner Notes..............

SECTION 41

Scanner::TriggerKey

The **TriggerKey** property assigns a hardware button to the scanner. After this association is made the hardware scanner will attempt a read operation each time the assigned button is pressed.

The devioTRIGGERKEY enumeration defines the values which can be assigned to the TriggerKey property. Each value corresponds to a CE *Application Launch Key*. By default the Application Launch Keys are typically assigned to operating system applications such as the Calendar and Contacts applications. If an application associates a launch key to the scanner using the TriggerKey property, the association will be removed when the Scanner object is destroyed or the TriggerKey property is set to **devioTRIGGERKEY_NONE**

For more information about Application Launch Keys refer to the CE or device manufacturer documentation.

```
Visual Basic Syntax
```

objScanner.TriggerKey = lngValue

See Also

Scanner::DataAvailabl
e ....
Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::OnDataAvail
able...............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM
ask................................
Scanner::WedgeMode...
Scanner Notes..............

SECTION 42

Scanner::WedgeMode

The **WedgeMode** property determines whether scanner input is sent directly to the active control on the active window. When this property is set to true, data read from a successful scan is sent to the control that has focus on the application's active window. If the property is set to false, scanned data is not sent directly to any window controls.

When wedge mode is activated, the window's focus should be on an input control such as a textbox or editable combo box. Controls that are not designed for text user input such as radio buttons or command buttons will not automatically display the data that is scanned during wedge mode. The OnDataAvailable (Scanner) event will continue to be triggered when the wedge mode is activated and the data scanned can be accessed using the GetData (Scanner) property.

Visual Basic Syntax

objScanner.WedgeMode = boolValue

See Also

Scanner::DataAvailabl
e ....
Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo
gyMask.........................
Scanner::OnDataAvail
able..............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM
ask...............................
Scanner::TriggerKey.....
Scanner Notes...............

SECTION 43

Scanner Notes

Symbol

Symbologies

The Symbol Scanner API supports a maximum of fifteen (15) simultaneously activated symbologies. Setting the symbology mask to devioBARCODE_ALL on Symbol scanners will enable the first 15 symbologies available to the scan engine. To ensure that the required symbologies for an application are enabled, it is best to explicitly activate the each desired symbology.

Casio

The Casio scanner API does not permit multiple applications to activate the scanner simultaneously.

Socket

See Also

Scanner::DataAvailabl e..................................
Scanner::Enabled..........
Scanner::GetData..........
Scanner::GetSymbolo gyMask........................
Scanner::OnDataAvail able..............................
Scanner::SoftTrigger.....
Scanner::Supported.......
Scanner::SymbologyM ask..................................
Scanner::TriggerKey.....
Scanner::WedgeMode...

The **Magnetic Stripe Reader (MSR)** object allows applications to communicate with readers that are built-in or attached to a device. Magnetic Stripe Readers take input from cards such as identification cards and credit cards that are equipped with stripes that contain data. The MSR object allows an application to set configuration properties of the stripe reader hardware as well as receive input when a card is scanned.

Some peripheral hardware such as printers may also be equipped with magnetic card readers. To utilize this type of hardware, the Printer object should be used.

The data read through the **MSR** object is accessible through the OnDataAvailable event. The actual text read by the **MSR** can be retrieved by calling the GetData . To set which character will be used to separate the card data fields, use the Separator property.

The Program ID for this object is **abdevio.MSR**.

Properties	Methods	Events
MSR::DataAvailable.....	MSR::GetData..............	MSR::OnDataAvailabl e ....
MSR::Enabled		
MSR::Separator		
MSR::Supported		
MSR::WedgeMode		

SECTION 44

MSR::DataAvailable

The **DataAvailable** property returns a boolean value indicating whether information has been retrieved from a magnetic stripe. If the value is true, data has been read from the stripe. If the value is false, no data has been read. If data has been read by the Magnetic Stripe Reader, the data can be accessed by calling the GetData (MSR) method.

Visual Basic Syntax

boolValue = objMSR.DataAvailable

See Also

MSR::Enabled ....
MSR::GetData ....
MSR::OnDataAvailabl
e ....
MSR::Separator ....
MSR::Supported ....
MSR::WedgeMode ....
MSR Notes ....

SECTION 45

MSR::Enabled

The **Enabled** property sets or retrieves the status of the Magnetic Stripe Reader object. If the value is set to true, the Magnetic Stripe Reader is activated. Upon activation the magnetic stripe reader settings are applied and the device is activated for asynchronous reading. If the value is set to false, asynchronous reading is terminated and the Magnetic Stripe Reader hardware is deactivated.

Visual Basic Syntax

objMSR.Enabled = boolValue

Error Values

ABERR_ERR_HWDLL	Error loading native MSR api.
ABERR_ENGINEINIT	Error initializing MSR engine.
ABERR_SETSEPARATOR	Unable to set MSR field separator.

See Also

MSR::DataAvailable.....
MSR::GetData.............
MSR::OnDataAvailabl
e.................................
MSR::Separator...........
MSR::Supported..........
MSR::WedgeMode.......
MSR Notes...................

SECTION 46

MSR::GetData

The **GetData** method returns the data held in the Magnetic Stripe Reader's input buffer. The MSR's reading buffer is filled with data each time a magnetic stripe is successfully read. This method returns a string value equivalent to the data retrieved from the tracks of the magnetic stripe. After this method is called, the data is removed from the MSR object's input buffer.

Visual Basic Syntax

strValue = objMSR.GetData

See Also

MSR::DataAvailable.....
MSR::Enabled ....
MSR::OnDataAvailabl
e ....
MSR::Separator ....
MSR::Supported ....
MSR::WedgeMode ....
MSR Notes ....

SECTION 47

MSR::OnDataAvailable

The **OnDataAvailable** event is fired when a read is completed successfully by the Magnetic Stripe Reader hardware. This event is only fired when the reader has read data successfully. If the hardware's attempt to read the stripe is not successful, the event will not fire since no data will be present. The procedure in your application that is associated with this event can be used to retrieve the data that was read using the GetData (MSR) method of the MSR object.

Visual Basic Syntax

```
Set objMSR = CreateObjectWithEvents("abdevio.MSR", "MSR_")
...
Sub MSR_OnDataAvailable()
...
End Sub
```

See Also

SECTION 48

MSR::Separator

The **Separator** property specifies an ASCII character that is the delimeter of the data read by the Magnetic Stripe Reader. The Separator property can be set to a numeric value equivalent to the ASCII character code of the desired delimeter or the property can be set to the actual delimeter character. The value returned by the GetData (MSR) method after a card is read will contain the separator character at the end of each track retrieved from the card.

Visual Basic Syntax

objMSR.Separator = varValue

Error Values

ABERR_INVALIDSEPARATOR	Invalid field separator applied.

See Also

MSR::DataAvailable.....
MSR::Enabled ....
MSR::GetData ....
MSR::OnDataAvailabl
e ....
MSR::Supported ....
MSR::WedgeMode.......
MSR Notes ....

SECTION 49

MSR::Supported

The **Supported** property returns a value indicating whether the currently installed library contains a complete Magnetic Stripe Reader object implementation. Certain devices supported by the Device Framework cannot be equipped with hardware Magnetic Stripe Readers and as a result the **abdevio** library that is automatically installed on the device is not capable of working with a Magnetic Stripe Reader. If this property returns true, the object contains an MSR implementation. If the property returns false, the object does not contain an MSR implementation and the MSR object should not be used.

This property does not verify if the API libraries or the MSR hardware is properly installed on the device.

Visual Basic Syntax

boolValue = objMSR.Supported

See Also

MSR::DataAvailable.....
MSR::Enabled.............
MSR::GetData.............
MSR::OnDataAvailabl
e..................................
MSR::Separator...........
MSR::WedgeMode.......
MSR Notes..................

SECTION 50

MSR::WedgeMode

The **WedgeMode** property determines whether Magnetic Stripe Reader input is sent directly to the active control on the active window. When this property is set to true, data read from a successful read is sent to the control that has focus on the application's active window. If the property is set to false, data is not sent directly to any window controls.

When wedge mode is activated, the window's focus should be on an input control such as a textbox or editable combo box. Controls that are not designed for text user input such as radio buttons or command buttons will not automatically display the data that is read during wedge mode. The OnDataAvailable (MSR) event will continue to be triggered when wedge mode is activated and the data read by the Magnetic Stripe Reader can be accessed using the GetData (MSR) property.

```
Visual Basic Syntax
```
objMSR.WedgeMode = boolValue

See Also

MSR::DataAvailable.....
MSR::Enabled ....
MSR::GetData.............
MSR::OnDataAvailabl
e ....
MSR::Separator............
MSR::Supported...........
MSR Notes ....

MSR Notes

CHAPTER 6

abImgCap (Image Capture)

The **abImgCap** library allows applications to make use of graphics files in a variety of formats.

The **SigCapture** object enables the display of images stored in files in any of the following formats: Graphics Interchange Format (GIF), Windows CE Bitmaps (BMP, 2BP), Portable Network Grapics format (PNG) and Joint Photographic Experts Group format (JPG). The SigCapture object is also capable of capturing stylus movements and saving the resulting images in JPG or BMP format.

The Program ID for this control is **abimgcap.SigCapture.**

Properties	Methods
SigCapture::Capture	SigCapture::Clear ....
SigCapture::HWindow..	SigCapture::Refresh ......
SigCapture::Image	SigCapture::SaveImag e....................................

SECTION 51

SigCapture::Capture

The **Capture** property sets the Signature Capture control into capture mode. While in capture mode, the application user may use the stylus to draw or write on the area of the screen occupied by the control. Any stylus movements captured by the control are rendered as black lines in the resulting image. Capture mode is enabled when this property is set to true and disabled when the property is set to false. If capture mode is disabled, the image displayed in the control may not be altered.

`Visual Basic Syntax`

objSigCapture.Capture = boolValue

See Also

SigCapture::Clear ....
SigCapture::HWindow..
SigCapture::Image........
SigCapture::Refresh......
SigCapture::SaveImag
e ....

SECTION 52

SigCapture::Clear

The **Clear** method discards the current image from the Signature Capture control. After the Clear method is called the control is displayed as a solid white rectangle. The **clear** method does not affect the source file from which an image was loaded.

```
Visual Basic Syntax
```

objSigCapture.Clear
See Also
SigCapture::Capture .....
SigCapture::HWindow..
SigCapture::Image........
SigCapture::Refresh......
SigCapture::SaveImag
e..................................

SECTION 53

SigCapture::HWindow

The **HWindow** property returns the handle of the window that makes up the visible area of the Signature Capture control. The handle obtained from this property can be used to perform direct manipulation of the control's visible area however caution should be exercised when accessing the control using this technique.

Visual Basic Syntax

lngValue = objSigCapture.HWindow
See Also
SigCapture::Capture .....
SigCapture::Clear ....
SigCapture::Image ....
SigCapture::Refresh......
SigCapture::SaveImag
e ....

SECTION 54

SigCapture::Image

The **Image** property allows you to load an image into the Signature Capture control. The property is a text string which must be set to the path and filename of an image file. When the property is set the Signature Capture control will immediately load the image. The supported images are CE Bitmaps (*.bmp*), Graphics Interchange Format (*.gif*), and Joint Photographic Experts Group format (*.jpg*). Once an image has been loaded, the **Image** property can also be used to retrieve the path and name of the file from which the image was loaded.

Visual Basic Syntax

```
objSigCapture.Image = strFileName
```

Error Values

ABERR_FILENOTFOUND	Image File was not found.
ABERR_LOADFAILED	Image could not be loaded.
ABERR_SIGNATURE	Expected file signature not found
ABERR_FORMAT_UNKNOWN	Sanity check failed.
ABERR_ACCESSDENIED	Access Denied.
ABERR_FORMAT_NOT_SUPPORTED	Format not Supported.
ABERR_INTERNAL	Internal Error.
ABERR_UNKNOWNTYPE	Unknown File Type.
ABERR_DIBTOOLARGE	Image Too Big.
ABERR_OUTOFMEMORY	Out of memory.
ABERR_EOF	Error end of File.

See Also

SigCapture::Capture .....
SigCapture::Clear ....
SigCapture::HWindow..
SigCapture::Refresh......
SigCapture::SaveImag

e ....

SECTION 55

SigCapture::Refresh

The **Refresh** method reloads the image shown in the Signature Capture control from the file currently specified by the Image property. If the image property is not set to a valid picture file the **Refresh** method will fail. This method can be used to rollback any changes made to the image while in capture mode.

```
Visual Basic Syntax
```

objSigCapture.Refresh

```
Error Values
```

ABERR_FILENOTFOUND	Image File was not found.
ABERR_LOADFAILED	Image could not be loaded.
ABERR_SIGNATURE	Expected file signature not found
ABERR_FORMATUNKNOWN	Sanity check failed.
ABERR_ACCESSDENIED	Access Denied.
ABERR_FORMAT_NOT_SUPPORTED	Format not Supported.
ABERR_INTERNAL	Internal Error.
ABERR_UNKNOWNTYPE	Unknown File Type.
ABERR_DIBTOOLARGE	Image Too Big.
ABERR_OUTOFMEMORY	Out of memory.
ABERR_EOF	Error end of File.

See Also

SigCapture::Capture .....
SigCapture::Clear ....
SigCapture::HWindow..
SigCapture::Image........
SigCapture::SaveImag
e ....

SECTION 56

SigCapture::SaveImage

The **SaveImage** method transfers the binary information comprising the current image to the specified file. The file is always saved in JPEG format so it is recommended to use the *.jpg* extension when specifying the filename. If the filename specifed already exists it will be overwritten by the new JPEG file created by this method.

Use the SaveImage method along with the Capture property to save signatures, markup etc... entered by a user using the Signature Capture control.

`Visual Basic Syntax`

objSigCapture.SaveImage Filename

`Error Values`

ABERR_SAVEFAILED	Could not save Image.

See Also

SigCapture::Capture .....
SigCapture::Clear ....
SigCapture::HWindow..
SigCapture::Image........
SigCapture::Refresh......

CHAPTER 7

abNotify (Power)

The **abnotify** library contains ojects which send notifications to applicaitons of system events. Some system events destroy resources that are in use by an active application and through the notfications sent by the abnotify library, an application is able to seamlessly handle re-creation of these resources in order to avoid operational errors.

The **Power** object sends notifications of power related events. The main power events are the *Power On* and *Power Off* events which occur when a device is turned on and off respectively.

Many hardware related application objects such as the Scanner object and RasConn object utilize hardware resources that are forcefully terminated when a device is powered down. The Power object can be used to signal to an application that these types of objects require initialization before continuing with normal operation of the application.

The Program ID for this control is **abnotify.Power**.

Properties	Events
Power::Enabled	Power::OnPowerOff......
	Power::OnPowerOn ......

SECTION 57

Power::Enabled

The **Enabled** property sets the notification state of the Power object. If the Enabled property is true, the Power object will fire notifications in response to power events. If the value is false, no notifications will be fired.

Visual Basic Syntax

objPower.Enabled = boolValue

Error Values

ABERR_THREADCREATE	An error ocurred while creating the notifi
ABERR_DRIVEROPEN	An error ocurred while opening the powe
ABERR_DRIVERCOMM	Failed to communicate with the power dr establish notification mechanism.
ABERR_REGISTERFAILED	Unable to register the power driver.
ABERR_MEMORYINIT	Unable to create shared memory used for initialization.
ABERR_MEMORYACCESS	Unable to access shared memory used for driver initialization.

See Also

Power::OnPowerOn......
Power::OnPowerOff.....

SECTION 58

Power::OnPowerOn

The **OnPowerOn** event is triggered when a device is turned on from a suspended state. The function registered by an application to handle this event can be used to check the status of and/or re-initialize resources which are in use by the application.

Visual Basic Syntax

```
Set objPower = CreateObjectWithEvents("abnotify.Power", "Power_")
...
Sub Power_OnPowerOn()
...
End Sub
```

See Also
Power::Enabled ....
Power::OnPowerOff .....

SECTION 59

Power::OnPowerOff

The **OnPowerOff** event is triggered when a device is turned off.

```
Visual Basic Syntax
Set objPower = CreateObjectWithEvents("abnotify.Power", "Power_")
...
Sub Power_OnPowerOff()
...
End Sub
```

See Also

Power::Enabled ....

Power::OnPowerOn......

CHAPTER 8

abR3Proxy (SAP Services)

The **abR3Proxy** component library provides a number of components that can be used to connect, execute RFCs, BAPIs and IDOCs in conjunction with the Pylon Connector for R/3.

In This Chapter

The **Connection** object is used to manage communications with a remote R/3 system. The properties of the Connection object are used to specify the parameters required to open a connection to an R/3 system and to modify the status of a connection once it has been opened. When using the abR3Proxy library an application is typically requried to create a Connection object whether operating in *online* or *offline* mode. The Connnection object is required whenever and Rfc , BAPI , or IDoc object is used to execute a call in *online* mode or queue a call while in *offline* mode.

The Program ID for this object is **AbR3Proxy.Connection**.

Properties	Methods
Connection::Client........	Connection::BeginTransaction........................
Connection::Connected..................................	Connection::Commit.....
Connection::Destination..................................	Connection::Connect.....
Connection::InTransaction..............................	Connection::Disconnect..................................
Connection::Language..	Connection::Rollback....
Connection::SoapObjectName........................	
Connection::Offline......	
Connection::Password...	
Connection::URL..........	
Connection::SoapProxy..................................	
Connection::User..........	

SECTION 60

Connection::BeginTransaction

The **BeginTransaction** method starts a BAPI transaction on the destination R/3 system. Transactions can be used to identify the start and end of a group of related calls such that the tasks performed by the calls can be reversed through the Rollback method or saved through the Commit method. After the BeginTransaction method is called, a transaction remains active until the Rollback or Commit methods are called. If the BeginTransaction method is called while a transaction is active an error is raised. It is good practice to commit or rollback any active transactions before closing the connection to R/3 through the Disconnect method.

The BeginTransaction method cannot be used while in *offline* mode (see Offline property). An application can insert the appropriate BAPIs into the asynchronous queue in the required sequence to create, commit and rollback transactions as necessary.

`Visual Basic Syntax`

objConnection.BeginTransaction

`Error Values`

ABERR_NOT_CONNECTED	A connection could not be established.
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is
ABERR_PREPARE_CALL_FAILED	Unable to format data to be transmitted
ABERR_NOTSUPPORTED_OFFLINE	No available in offline mode.
ABERR_TRANSACTION_ALREADYSTARTED	A transaction is already active on the con

See Also

SECTION 61

Connection::Client

The **Client** property sets or gets the R/3 client number to be used when establishing a connection to a system. The Client property is a string value holding the client number. When the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called, the Client value stored in the connection object is used to determine the system where the connection will be established.

`Visual Basic Syntax`

objConnection.Client = strValue

See Also

SECTION 62

Connection::Commit

The **Commit** method completes a BAPI transaction on the destination R/3 system. A transaction can be started with the BeginTransaction method. When a transaction is commited through the Commit method, the changes to the R/3 system, performed by any BAPI called since the transaction was started, are saved. After the Commit method is called the transaction is ended and a new transaction must be started before the Commit method is called again. If a transaction is not active when this method is called an error will be raised.

The Commit method cannot be used while in *offline* mode (see Offline property). An application can insert the appropriate BAPIs into the asynchronous queue in the required sequence to create, commit and rollback transactions as necessary.

```
Visual Basic Syntax
```

objConnection.Commit

```
Error Values
```

ABERR_NOT_CONNECTED	A connection could not be established.
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is
ABERR_PREPARE_CALL_FAILED	Unable to format data to be transmitted
ABERR_NOTSUPPORTED_OFFLINE	No available in offline mode.

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Connect ....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName........................
Connection::Offline......
Connection::Password...
Connection::Rollback ...
Connection::URL..........
Connection::SoapProx
y..................................
Connection::User..........

SECTION 63

Connection::Connect

The **Connect** method is used to open a connection to an R/3 system. When the Connect method is called, the Connection object uses the properties assigned to the object to establish a connection with an R/3 system. Most of the properties of the Connection object must be set before the Connect method is called. If a required property has not been assigned a value an error is raised.

When a connection is successfully opened it is possble to make *online* calls to the R/3 system specified in the Destination property.

Visual Basic Syntax

objConnection.Connect

```
Error Values
```

ABERR_ALREADY_CONNECTED	The connection is already open.
ABERR_URL_NOT_SPECIFIED	The URL to the SOAP router has not been se
ABERR_OBJECTNAME_NOT_SPECIFIED	The object name property is empty.
ABERR_USER_NOT_SPECIFIED	The user property is empty.
ABERR_PASSWORD_NOT_SPECIFIED	The password property is empty.
ABERR_CLIENT_NOT_SPECIFIED	The client property is empty.
ABERR_DESTINATION_NOT_SPECIFIED	The destination property is empty.
ABERR_LANGUAGE_NOT_SPECIFIED	The language property is empty.
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or is una\
ABERR_PREPARE_CALL_FAILED	Unable to format data to be transmitted
ABERR_NOTSUPPORTED_OFFLINE	Not available in offline mode
ABERR_CONNECTION_FAILED	The server denied access or the connection ir not correct.

See Also

Connection::BeginTransaction........................
Connection::Client........
Connection::Commit.....
Connection::Connected..................................
Connection::Destination..................................
Connection::Disconnect..................................
Connection::InTransaction..............................
Connection::Language..
Connection::SoapObjectName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback...

SECTION 64

Connection::Connected

The **Connected** property is a read-only property that returns a boolean value indicating the connectivity status of the Connection object. If the value is **true**, a connection is currently open with an R/3 system. If the value is **false**, there is no connection open. A connection can be opened using the Connect method.

Visual Basic Syntax

boolValue = objConnection.Connected

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback ...
Connection::URL..........
Connection::SoapProx
y..................................
Connection::User..........

SECTION 65

Connection::Destination

The **Destination** property sets or gets the connection string that determines the location of an R/3 system. The Destination property is a string value holding either the name of a *saved configuration* or a full connection string. A *saved configuration* can be created in administration utility of the Atoma Connector. When a *saved configuration* is created the properties specifying the destination are entered and saved under a name that can be assigned to the Destination property. When the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called, the destination stored in the connection object is used to determine the network location of the R/3 system.

`Visual Basic Syntax`

objConnection.Destination = strValue
See Also
Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect....
Connection::Connecte
d..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback...
Connection::URL..........
Connection::SoapProx
y..................................
Connection::User..........

SECTION 66

Connection::Disconnect

The **Disconnect** method closes a connection to an R/3 system. A connection to can be opened using the Connect method. Any pending transactions should be ended before a connection is closed. A transaction can be ended with either the Commit or Rollback method depending on the desired outcome. If a connection is closed before a pending transaction is ended errors may occur.

Visual Basic Syntax

objConnection.Disconnect

```
Error Values
```

ABERR_URL_NOT_SPECIFIED	The URL to the SOAP router has not been se
ABERR_OBJECTNAME_NOT_SPECIFIED	The object name property is empty.
ABERR_NOT_CONNECTED	A connection has not been established.
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or is una\
ABERR_PREPARE_CALL_FAILED	Unable to format data to be transmitted
ABERR_NOTSUPPORTED_OFFLINE	Not available in offline mode

See Also

Connection::BeginTransaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect....
Connection::Connected..................................
Connection::Destination..................................
Connection::InTransaction..............................
Connection::Language..
Connection::SoapObjectName........................
Connection::Offline......
Connection::Password...
Connection::Rollback...
Connection::URL..........
Connection::SoapProxy..................................
Connection::User..........

SECTION 67

Connection::InTransaction

The **InTransaction** property is a read-only property that returns a boolean value indicating the transactional status of the Connection object. If the value is **true**, a transaction is currently active. If the value is **false**, there is no transaction active. Any BAPI calls performed while a transaction is active can be reversed using the Rollback method or saved using the Commit method. A transaction can be started with the BeginTransaction method.

Visual Basic Syntax

boolValue = objConnection.InTransaction

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::Language..
Connection::SoapObje
ctName........................
Connection::Offline......
Connection::Password...
Connection::Rollback ...
Connection::URL..........
Connection::SoapProx
y..................................
Connection::User..........

S E C T I O N 6 8

Connection::Language

The **Language** property sets or gets the R/3 language code selected during a connection to an R/3 system. The Language property is a string value holding the appropriate language code. When the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called, the logon information stored in the connection object is used to establish a connection to R/3. The language selected may affect the formatting of values returned from calls made to the system. Consult your R/3 system and/or documentation for more information on the supported language codes.

`Visual Basic Syntax`

objConnection.Language = strValue
See Also

SECTION 69

Connection::SoapObjectName

The **ObjectName** property sets or gets the name of the R/3 Soap Proxy object as it is registered in the router specified in the RouterURL property. The default value of this property is *soap-sap-connector*. The abR3Proxy library uses the SOAP protocol to connect and communicate with an R/3 system in conjunction with an R/3 Soap Proxy object. The registration name of the R/3 Soap Proxy object is normally specified when the R/3 Soap Proxy is installed on a machine and will ordinarily be equivalent to the default value.

Visual Basic Syntax

objConnection.ObjectName = strValue

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::Offline......
Connection::Password...
Connection::Rollback...
Connection::URL..........
Connection::SoapProx
y..................................
Connection::User..........

SECTION 70

Connection::Offline

The **Offline** property is a read/write property used to specify the mode of operation desired while working with the Connection object. The offline property is a boolean value indicating the assigned mode of oepration. The connection property can be used in two modes: *online* and *offline*. When the Offline property is set to **true**, *offline* mode is activated. When the property is set to **false** *online* mode is activated. When *offline* mode is activated, the Connection object will not attempt to contact a remote system when the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called. This can provide a significant performance increase for an application using the Connection object when it is known that there is no network connection available. When *online* mode is activated the Connection object will try to contact remote systems when necessary.

Visual Basic Syntax

objConnection.Offline = boolValue

See Also

SECTION 71

Connection::Password

The **Password** property sets or gets the user password of the R/3 User to be used to authenticate a connection to an R/3 system. The Password property is a string value holding a user's password. When the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called, the logon information stored in the connection object is used to establish a connection to R/3.

Visual Basic Syntax

objConnection.Password = strValue

See Also

Connection::BeginTra
nsaction ....
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Connecte
d ....
Connection::Destinatio
n ....
Connection::Disconne
ct ....
Connection::InTransac
tion ....
Connection::Language ..
Connection::SoapObje
ctName ....
Connection::Offline ......
Connection::Rollback ...
Connection::URL..........
Connection::SoapProx
y ....
Connection::User..........

SECTION 72

Connection::Rollback

The **Rollback** method terminates a BAPI transaction on the destination R/3 system. A transaction can be started with the BeginTransaction method. When a transaction is ended through the Rollback method, the changes to the R/3 system, performed by any BAPI called since the transaction was started, are reversed. After the Rollback method is called the transaction is ended and a new transaction must be started before the Rollback method is called again. If a transaction is not active when this method is called an error will be raised.

The Rollback method cannot be used while in *offline* mode (see Offline property). An application can insert the appropriate BAPIs into the asynchronous queue in the required sequence to create, commit and rollback transactions as necessary.

`Visual Basic Syntax`

objConnection.Rollback

`Error Values`

ABERR_NOT_CONNECTED	A connection could not be established.
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is
ABERR_PREPARE_CALL_FAILED	Unable to format data to be transmitted
ABERR_NOTSUPPORTED_OFFLINE	No available in offline mode.

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName.........................
Connection::Offline......
Connection::Password...
Connection::URL.........
Connection::SoapProx
y..................................
Connection::User..........

SECTION 73

Connection::URL

The **URL** property sets or gets the network address of a SOAP router. The URL property is a string value holding a Uniform Resource Locator (URL) corresponding to a SOAP router. The abR3Proxy library uses the SOAP protocol to connect and communicate with an R/3 system in conjunction with an R/3 Soap Proxy object. The R/3 Soap Proxy object should be registered in the router assigned to the Connection object throught the URL property.

`Visual Basic Syntax`

objConnection.RouterURL = strValue

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback ...
Connection::SoapProx
y..................................
Connection::User..........

SECTION 74

Connection::SoapProxy

The **SoapProxy** property is a read-only property that returns a reference to an object that manages the underlying communications of the abR3Proxy library. The object returned is an instance of the SoapProxy object exposed by the absoap library. The object returned by the SoapProxy property exposes the following methods and properties FaultActor , FaultCode , FaultDetail FaultString , HttpRequestResult, HttpStatus , ResponseText , ResponseXML , Result , and SoapFaultExists . The HttpRequestResult property returns an HRESULT value indicating the status of the last call executed by the object.

Visual Basic Syntax

Set objSoap = objConnection.SoapProxy

See Also

Connection::BeginTransaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect ....
Connection::Connected..................................
Connection::Destination..................................
Connection::Disconnect.................................
Connection::InTransaction..............................
Connection::Language..
Connection::SoapObjectName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback ...
Connection::URL..........
Connection::User..........

SECTION 75

Connection::User

The **User** property sets or gets the logon ID of the R/3 User to be used to authenticate a connection to an R/3 system. The User property is a string value holding a user logon ID. When the Connect method is called or the Execute and Post methods of a BAPI , Rfc and IDoc objects are called, the logon information stored in the connection object is used to establish a connection to R/3. The user specified here should have the appropriate rights on the system to perform the business tasks associated with any RFC, BAPI or IDoc called using the Connection object.

Visual Basic Syntax

objConnection.User = strValue

See Also

Connection::BeginTra
nsaction........................
Connection::Client........
Connection::Commit.....
Connection::Connect....
Connection::Connecte
d..................................
Connection::Destinatio
n..................................
Connection::Disconne
ct.................................
Connection::InTransac
tion..............................
Connection::Language..
Connection::SoapObje
ctName.........................
Connection::Offline......
Connection::Password...
Connection::Rollback...
Connection::URL..........
Connection::SoapProx
y...................................

SECTION 76

The **IDoc** object is used to manage SAP R/3 *Intermediate Documents* (IDocs). Use the IDoc object whenever posting IDocs to R/3.

The Program ID for this object is **AbR3Proxy.IDoc**.

Properties	Methods
IDoc::Extension............	IDoc::AddSegment........
IDoc::IDocType............	IDoc::LoadRequest........
IDoc::MessageType......	IDoc::LoadResponse.....
	IDoc::Post.....................
	IDoc::Queue..................

SECTION 77

IDoc::AddSegment

The **AddSegment** method is used to append a *first-level* segment to an IDoc object. The AddSegment method takes one string parameter indicating the segment type name. If the segment is successfully added the method returns a reference to the new Segment object. A first-level segment is directly associated with the root of the IDoc tree structure and does not have a parent segment. Consult your system and/or R/3 documentation for more information on available IDocs and their corresponding segments.

`Visual Basic Syntax`

Set objSegment = objIdoc.AddSegment(strValue)

See Also

IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 78

IDoc::Name

The **Name** property is a read/write property that sets the IDoc type of an IDoc. Each IDoc that is registered on an R/3 system is assigned an IDoc type or name. Examples of common IDoc types are "WMMBID01" and "MATMAS02". When a new IDoc is created through extension, it typically has a *basic IDoc type* that identifies the IDoc structure that was extended to create the new IDoc. The basic IDoc type can be set using the IDocType property. Consult your system and/or R/3 documentation for more information on available IDocs and their corresponding IDoc types.

Visual Basic Syntax

objIdoc.Name = strValue

See Also

IDoc::AddSegment ....
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 79

IDoc::Extension

The **Extension** property is a read/write property that represents the extension name of the IDoc .

Visual Basic Syntax

```
objIDoc.Extension = strValue
```

See Also

IDoc::AddSegment ....
IDoc::Name ....
IDoc::IDocType............
IDoc::LoadRequest ....
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue ....
IDoc::Release ....
IDoc::SegmentCount ....
IDoc::Segments ....
IDoc::Sender.................
IDoc::TransactionID .....

SECTION 80

IDoc::IDocType

The **IDocType** property is a read/write property that sets the basic IDoc type of an extended IDoc. An extended IDoc is typically based on one of the default IDoc types delivered with a standard R/3 system. This parameter is optional and is usually only necessary when creating a custom IDoc structure that has been extended from a standard IDoc.

```
Visual Basic Syntax
```

objIdoc.IDocType = strValue

See Also

IDoc::AddSegment ....
IDoc::Name ....
IDoc::Extension ....
IDoc::LoadRequest ....
IDoc::LoadResponse.....
IDoc::MessageType ......
IDoc::Post.....................
IDoc::Queue ....
IDoc::Release ....
IDoc::SegmentCount ....
IDoc::Segments ....
IDoc::Sender.................
IDoc::TransactionID .....

SECTION 81

IDoc::LoadRequest

The **LoadRequest** method recreates a previously created IDoc from a SOAP envelope. The LoadRequest method takes one string parameter that is used to pass the SOAP envelope as text to the method. The IDoc object uses the envelope to build the necessary segment and field objects associated with the IDoc such that they can be accessed using the methods and properties of the object. Typically the AsyncPost object is used along with the LoadRequest method to recreate an IDoc that was transmitted asynchronously using the Queue method.

Visual Basic Syntax

objIdoc.LoadRequest strValue

Error Values

ABERR_SOAPPROXYCREATE_FAILED	Unable to create SOAP proxy object.
ABERR_PARSEREQUEST_FAILED	Could not interpret envelope.

See Also

IDoc::AddSegment.......
IDoc::Name.................
IDoc::Extension...........
IDoc::IDocType...........
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post....................
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments ....
IDoc::Sender................
IDoc::TransactionID.....

SECTION 82

IDoc::LoadResponse

The **LoadResponse** method imports the results of an IDoc transmission into n IDoc object. The LoadResponse method takes one string parameter that is used to pass a SOAP envelope as text to the method. The IDoc object uses the envelope to retrieve the results of the IDoc submission so that the transaction id returned can be accessed using the TransactionID property of the object. Typically the AsyncPost object is used along with the LoadResponse method to handle a response returned from an IDoc that was transmitted asynchronously using the Queue method.

`Visual Basic Syntax`

objIdoc.LoadResponse strValue

`Error Values`

ABERR_CREATE_XML_FAILED	Unable to load response data.
ABERR_PARSE_FAILED	Unable to interpret return data.
ABERR_IDOC_POST_FAILED	R/3 function returned an error.

See Also

IDoc::AddSegment.......
IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 83

IDoc::MessageType

The **MessageType** property is a read/write property that sets the Application Link Enabling (ALE) message type associated to an IDoc. Each IDoc is registered on an R/3 system a message type is typically associated with the IDoc. Consult your system and/or R/3 documentation for more information on available IDocs and their corresponding message types.

`Visual Basic Syntax`

```
objIdoc.MessageType = strValue
```

See Also

IDoc::AddSegment ....
IDoc::Name ....
IDoc::Extension ....
IDoc::IDocType............
IDoc::LoadRequest ....
IDoc::LoadResponse.....
IDoc::Post.....................
IDoc::Queue ....
IDoc::Release ....
IDoc::SegmentCount ....
IDoc::Segments ....
IDoc::Sender.................
IDoc::TransactionID .....

SECTION 84

IDoc::Post

The **Post** method is used to transmit the IDoc specified by the IDoc object to an R/3 system. The Post method takes one parameter, an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to deliver the IDoc to a system. When the Post method is called, the IDoc object attempts to connect to the R/3 system identified by the connection object and to submit the IDoc, as defined by the current properties of the IDoc object.

The IDoc is submitted to the Application Link Enabling (ALE) layer of the the R/3 system specified by the Connection object. The result returned by the Post method does not indicate suscess or failure of the functional/business tasks performed by the IDoc but instead only indicates whether the R/3 system accepted the new IDoc and added it to its transactional queue. When an IDoc is sucessfully submitted to an R/3 system the TransactionID property of the IDoc object is filled with the transaction Id returned from R/3. The IDoc object executes the necessary calls using the SOAP protocol and attempts to connect to a SOAP router where an R/3 proxy object is registered.

`Visual Basic Syntax`

objIdoc.Post objConnection

`Error Values`

ABERR_CONNECTION_FAILED	Could not connect to R/3
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is una
ABERR_SOAP_PREPARE_FAILED	Could not create call envelope.
ABERR_NAME_NOTSPECIFIED	Name of RFC is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECIFIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIED	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIED	Language specifier not assigned to connection

See Also

IDoc::AddSegment.......
IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 85

IDoc::Queue

The **Queue** method is used to submit an IDoc to the AsyncPost object. When an IDoc is submitted to the AsyncPost object it is placed in a queue of asynchronous calls that will be executed when a Tx-Sync operation is performed by the user. The results obtained from the submission of an IDoc to an R/3 system are stored in the AsyncPost object after the Tx-Sync operation.

The Queue method takes three parameters. The first parameter is an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to complete the call. The second parameter, the *name* parameter, is a string value used to identify the queue of asynchonous calls associated with the application. The third parameter, the *id* parameter, is a string value used to assign an custom identifier to the asynchronous call.

The *name* and *id* parameters can be used to retrieve the results of an IDoc submission from the AsyncPost object after the IDoc has been transmitted during a Tx-Sync operation. When the payload of the response from the call is loaded into the IDoc object through the LoadResponse method, the transaction id (if returned) will be available via the TransactionID property.

`Visual Basic Syntax`

objIdoc.Queue objConnection, strName, strId

`Error Values`

ABERR_NAME_NOTSPECIFIED	Name of the RFC is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECIFIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIED	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIED	Language specifier not assigned to connection

See Also

IDoc::AddSegment.......
IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post....................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 86

IDoc::Release

The **Release** property is a read/write property that sets the version of the IDoc structure used by the IDoc object. This property is a string value that is typically equivalent to the release number of the R/3 system where the IDoc structure is defined. Examples of common release numbers are "45B" and "46C".

Visual Basic Syntax

```
objIdoc.Release = strValue
```

See Also

IDoc::AddSegment.......
IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue.................
IDoc::SegmentCount....
IDoc::Segments............
IDoc::Sender.................
IDoc::TransactionID.....

SECTION 87

IDoc::SegmentCount

The **SegmentCount** property returns the total number of *first-level* segments currently associated with an IDoc object. A first-level segment is directly associated with the root of the IDoc tree structure and does not have a parent segment. A segment can be added to the IDoc object using the AddSegment method.

Visual Basic Syntax

```
lngValue = objIdoc.SegmentCount
```

See Also

IDoc::AddSegment ....
IDoc::Name ....
IDoc::Extension ....
IDoc::IDocType ....
IDoc::LoadRequest ....
IDoc::LoadResponse .....
IDoc::MessageType ......
IDoc::Post ....
IDoc::Queue ....
IDoc::Release ....
IDoc::Segments ....
IDoc::Sender ....
IDoc::TransactionID .....

SECTION 88

IDoc::Segments

The **Segments** property is a read-only property that returns an instance of a Segment object. The Segments property is used to retrieve a *first-level* segment associated with an IDoc object. The Segments property takes one string parameter that is used to indicate the segment type name of the desired first-level segment. If the segment specified does not exist an error is raised.

First-level segments are associated directly with the root of the IDoc tree structure and do not have any parent segment. The AddSegment method of the IDoc object can be used to create first-level segments.

Visual Basic Syntax

Set objSegment = objIDoc.Segments(strValue)

Error Values

ABERR_SEGMENT_NOT_FOUND	The segment specified was not found.

See Also

IDoc::AddSegment.......
IDoc::Name..................
IDoc::Extension............
IDoc::IDocType............
IDoc::LoadRequest.......
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue.................
IDoc::Release...............
IDoc::SegmentCount....
IDoc::Sender................
IDoc::TransactionID.....

SECTION 89

IDoc::Sender

The **Sender** property is a read/write property that identifies an application where an IDoc originates. The value assigned to the Sender property must be listed as an ALE partner in the destination R/3 system. Partners can be created and configured through the ALE configuration transactions on an R/3 system. Also note that the IDocs passed between an ALE partner and an R/3 system typically need to be identified and defined in the R/3 system before the partner/application attempts to send or receive IDocs with the R/3 system.

`Visual Basic Syntax`

strValue = objIDoc.Sender

See Also

IDoc::AddSegment ....
IDoc::Name ....
IDoc::Extension ....
IDoc::IDocType............
IDoc::LoadRequest ....
IDoc::LoadResponse.....
IDoc::MessageType......
IDoc::Post.....................
IDoc::Queue ....
IDoc::Release ....
IDoc::SegmentCount ....
IDoc::Segments ....
IDoc::TransactionID .....

Section 90

IDoc::TransactionID

The **TransactionID** is a read/write property that holds a unique identifier (also *TID*) assigned to an IDoc. When an R/3 system acknowledges receipt of an inbound IDoc it returns the transaction id assigned to the IDoc. This assigned transaction id can be retrieved from the TransactionID property after the IDoc is sent using the Post method. After an IDoc has be posted once, it can be posted again as a new transaction by clearing the TransactionID property and re-posting the IDoc.

```
Visual Basic Syntax
```

strValue = objIdoc.TransactionID

See Also

BAPI

The **BAPI** object is used to interact with an SAP R/3 system's Business Application Programming Interface (BAPI) objects. The BAPI object allows you to specify the BAPI being called by name and to add any desired simple parameters, structure parameters and inbound/outbound tables associated with the BAPI.

The Program ID for this object is **abR3Proxy.BAPI**.

Properties	Methods
BAPI::InTableCount.....	BAPI::ExecuteFunctio
BAPI::InTables.............	n....................................
BAPI::Name.................	BAPI::LoadRequest ......
BAPI::OutTableCount ..	BAPI::LoadResponse ....
BAPI::OutTables	BAPI::Queue.................
BAPI::Params...............	
BAPI::Params...............	
BAPI::ResultCount.......	
BAPI::Results...............	

SECTION 91

BAPI::ExecuteFunction

The **Execute** method is used to perform a BAPI call created using the BAPI object. The Execute method takes two parameters. The first parameter is an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to complete the call. The second parameter is a boolean value indication wheter a BAPI commit should be performed after the call is completed. When the value is **true** a commit call will be attempted after the BAPI call is successfully completed. If the value is **false** a commit call will not be attempted.

When the Execute method is called, the BAPI object attempts to connect to the R/3 system identified by the connection object and to perform the BAPI call according the current properties of the BAPI object. The BAPI object executes the necessary calls using the SOAP protocol and attempts to connect to a SOAP router where an R/3 proxy object is registered.

```
Visual Basic Syntax
```

objBAPI.Execute objConnection, boolValue

```
Error Values
```

ABERR_CONNECTION_FAILED	Could not connect to R/3
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is una
ABERR_SOAP_PREPARE_FAILED	Could not create call envelope.
ABERR_NAME_NOTSPECIFIED	Name of BAPI is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECIFIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIED	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIED	Language specifier not assigned to connection

See Also

BAPI::InTables............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params..............
BAPI::Params..............
BAPI::Queue...............
BAPI::ResultCount.......
BAPI::Results..............

SECTION 92

BAPI::InTables

The **InTables** property can be used to to create an inbound table or to retrieve a previously created inbound table. This property takes one string parameter that specifies the name of the table desired. The InTables property returns a reference to a Table object that can be used to add fields and values to the inbound table. If the table, specified in the parameter to the InTables property, does not exist it is created.

`Visual Basic Syntax`

objTable = objBAPI.InTables(strName)

`Error Values`

ABERR_TABLE_NOTFOUND	Unable to retrieve named table
ABERR_TABLECREATE_FAILED	Could not create table parameter.

See Also

BAPI::ExecuteFunctio
n..................................
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params..............
BAPI::Params..............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results..............

Section 93

BAPI::InTableCount

The **InTableCount** property returns the total number of inbound tables currently associated with a BAPI object. An inbound table can be added to the BAPI object using the InTables property.

```
Visual Basic Syntax
```

lngValue = objBAPI.InTableCount

See Also

BAPI::ExecuteFunctio
n..................................
BAPI::InTables.............
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name.................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params...............
BAPI::Params...............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results...............

Section 94

BAPI::LoadRequest

The **LoadRequest** method recreates a previously created BAPI call from a SOAP envelope. The LoadRequest method takes one string parameter that is used to pass the SOAP envelope as text to the method. The BAPI object uses the envelope to build the necessary table, parameter and field objects associated with the call such that they can be accessed using the methods and properties of the object. Typically the AsyncPost object is used along with the LoadRequest method to recreate a BAPI call that was called asynchronously using the Queue method.

Visual Basic Syntax

objBAPI.LoadRequest strValue

Error Values

ABERR_SOAPPROXYCREATE_FAILED	Unable to create SOAP proxy object.
ABERR_PARSEREQUEST_FAILED	Could not interpret envelope.

See Also

BAPI::ExecuteFunctio
n..................................
BAPI::InTables............
BAPI::InTableCount.....
BAPI::LoadResponse....
BAPI::Name................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params...............
BAPI::Params...............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results...............

SECTION 95

BAPI::LoadResponse

The **LoadResponse** method imports the results of a BAPI call into a BAPI object. The LoadResponse method takes one string parameter that is used to pass a SOAP envelope as text to the method. The BAPI object uses the envelope to build the necessary result parameter and outbound table objects returned by the call such that they can be accessed using the methods and properties of the object. Typically the AsyncPost object is used along with the LoadResponse method to handle a response returned from a BAPI call that was called asynchronously using the Queue method.

Visual Basic Syntax

objBAPI.LoadResponse strValue

Error Values

ABERR_SOAPPROXYCREATE_FAILED	Unable to create SOAP proxy object.
ABERR_PARSEREQUEST_FAILED	Could not interpret envelope.
ABERR_FUNCTIONEXECUTE_FAILED	R/3 function returned an error.

See Also

SECTION 96

BAPI::Name

The **Name** property is a read/write property that holds the text used to refer to the BAPI associated with a BAPI object. The name property is a string that must be assigned before the Execute or Queue methods are called. The text assigned to the Name property should be equivalent to the name of a valid BAPI active on the destination R/3 system. Consult you SAP system and/or documentation for more information on BAPI names.

Visual Basic Syntax

objBAPI.Name = strValue

See Also

BAPI::ExecuteFunctio
n..................................
BAPI::InTables.............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params...............
BAPI::Params...............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results...............

SECTION 97

BAPI::OutTableCount

The **OutTableCount** property returns the total number of outbound tables currently associated with a BAPI object. An outbound table can be retrieved from the BAPI object using the OutTables property.

Visual Basic Syntax

lngValue = objBAPI.OutTableCount
See Also
BAPI::ExecuteFunctio
n..................................
BAPI::InTables.............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name.................
BAPI::OutTables..........
BAPI::Params...............
BAPI::Params...............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results...............

SECTION 98

BAPI::OutTables

The **OutTables** property is used to retrieve an outbound table returned by a BAPI. This property takes one string parameter that specifies the name of the table desired. The OutTables property returns a reference to a Table object that can be used to access the fields and values contained in the table. If the table, specified in the parameter to the OutTables property, does not exist an error occurs. The Execute or LoadResponse method should be successfully called before attempting to access the outbound tables returned by an BAPI.

Visual Basic Syntax

Set objTable = objBAPI.OutTables(strName)

Error Values

ABERR_TABLE_NOTFOUND	Unable to retrieve named table

See Also

SECTION 99

BAPI::Params

The **ParamCount** property returns the total number of parameters currently associated with a BAPI object. A parameter can be added to the BAPI object using the Params property.

```
Visual Basic Syntax
```

lngValue = objBAPI.ParamCount
See Also
BAPI::ExecuteFunctio
n...................................
BAPI::InTables.............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name.................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params...............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results..............

SECTION 100

BAPI::Params

The **Params** property is used to create or retrieve an argument that will be passed to a BAPI call. The Params property takes one string parameter that identifies an arguement associated with a BAPI call. If the parameter does not exist it is created when the params property is called. This property returns a reference to a Param object. The methods and properties of the Param object can be used to add values to the BAPI parameter.

Visual Basic Syntax

Set objParam = objBAPI.Params(strName)

Error Values

ABERR_PARAMETER_NOTFOUND	Parameter could not be retrieved.
ABERR_CREATEPARAMETER_FAILED	Unable to create new parameter.

See Also

BAPI::ExecuteFunctio
n..................................
BAPI::InTables............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params..............
BAPI::Queue................
BAPI::ResultCount.......
BAPI::Results..............

SECTION 101

BAPI::Queue

The **Queue** method is used to submit a BAPI call to the AsyncPost object. When a BAPI call is submitted to the AsyncPost object it is placed in a queue of asynchronous function calls that will be executed when a Tx-Sync operation is performed by the user. The response returned from the BAPI called is stored in the AsyncPost object after the Tx-Sync operation.

The Queue method takes three parameters. The first parameter is an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to complete the call. The second parameter, the *name* parameter, is a string value used to identify the queue of asynchonous calls associated with the application. The third parameter, the *id* parameter, is a string value used to assign an custom identifier to the asynchronous call. The *name* and *id* parameters can be used to retrieve the response to the BAPI call from the AsyncPost object after the call has been executed during a Tx-Sync operation.

`Visual Basic Syntax`

objBAPI.Queue objConnection, strName, strId

`Error Values`

ABERR_NAME_NOTSPECIFIED	Name of BAPI call is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECIFIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIED	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIED	Language specifier not assigned to connection

See Also

BAPI::ExecuteFunction..................................
BAPI::InTables.............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name.................
BAPI::OutTableCount ..
BAPI::OutTables ....
BAPI::Params...............
BAPI::Params...............
BAPI::ResultCount.......
BAPI::Results...............

SECTION 102

BAPI::ResultCount

The **ResultCount** property returns the total number of return parameters currently associated with a BAPI object. A return parameter can be retrieved from the BAPI object using the Results property.

```
Visual Basic Syntax
```

lngValue = objBAPI.ResultCount
See Also
BAPI::ExecuteFunctio
n.................................
BAPI::InTables............
BAPI::InTableCount.....
BAPI::LoadRequest......
BAPI::LoadResponse....
BAPI::Name................
BAPI::OutTableCount..
BAPI::OutTables..........
BAPI::Params..............
BAPI::Params..............
BAPI::Queue................
BAPI::Results..............

SECTION 103

BAPI::Results

The **Results** property is a read-only property that returns a Param object corresponding to an argument returned by a BAPI Call. This property takes one string value corresponding to the name of the return parameter desired. Return parameters are only available after the Execute or LoadResponse methods have been successfully called. Consult your SAP system and/or documentation for the names of available BAPI objects and associated return parameters.

Visual Basic Syntax

```
Set objParam = objBAPI.Results( strValue )
```

Error Values

ABERR_PARAMETER_NOTFOUND	Parameter could not be retrieved.

See Also

Field

The **Field** object holds the value of an item in a structure parameter. The Param object is used to create and retrieve field items in a structure so that is never necessary to manually create and instance of the Field object.

The ProgID for this object is **abR3Proxy.Field**.

Properties

Field::Name..................
Field::Value..................

Section 104

Field::Name

The **Name** property is a read/write property that specifies the text used to refer to a field in a structure. Consult your SAP R/3 documentation to determine the appropriate name of a field associated with and Rfc or a BAPI structure.

`Visual Basic Syntax`

objField.Name = strValue
See Also
Field::Value..................

SECTION 105

Field::Value

The **Value** property can be used to set or retrieve the data that will be passed to a field in a structure parameter. The value set using the Value property should be a simple Variant type such as a float or string.

```
Visual Basic Syntax
```

objField.Value = varValue
See Also
Field::Name..................

The **Param** object is used to set the values and field names associated with an Rfc or BAPI call. The Rfc and BAPI objects provide properties that allow the creation and retrieval of Param objects corresponding to the associated parameters. Thus it is generally not necessary to manually create instances of the Param object.

There are two types of parameters represented by the Param object: *simple* and *structure* parameters. A *simple* parameter is assigned a single value such as an integer or string. A *structure* parameter consists of multiple Fields where each field can be assigned a single value.

The Program ID for this object is **abR3Proxy.Param.**

Properties

Param::FieldCount........
Param::Field.................
Param::IsSimple...........
Param::Name................
Param::Value................

SECTION 106

Param::Field

The **Fields** property is a read/write property that is used to set and retrieve values in a structure parameter. The Fields property takes one parameter that specifies the name or index number of the field. When the name of the desired field is passed to the Fields property as a string, the value of the named field is set or retrieved directly. When the index number of the field is passed to the Fields property, a reference to a Field Object is returned.

Visual Basic Syntax

objParam.Fields(strName) = strValue
Set objField = objParam.Fields(intIndex)

Error Values

ABERR_PROPERTY_NOT_SUPPORTED	Property is not valid on this instance of the ob

See Also

Param::FieldCount........
Param::IsSimple ....
Param::Name................
Param::Value................

Section 107

Param::FieldCount

The FieldCount property returns the total number of fields currently associated with a structure. A field can be added to a structure parameter using the Fields property.

```
Visual Basic Syntax
```

lngValue = objParam.FieldCount
See Also
Param::Field ....
Param::IsSimple ....
Param::Name................
Param::Value................

SECTION 108

Param::IsSimple

The **IsSimple** property ir a read only property that can be used to determine if a Param object refers to a structure or a single value. If the Param object is a simple parameter (single value) the IsSimple property will return true and the object does not support the Fields property. If the Param object is a complex parameter (structure) the IsSimple property returns **false**.

```
Visual Basic Syntax
```

boolValue = objParam.IsSimple
See Also
Param::Field.................
Param::FieldCount........
Param::Name................
Param::Value................

SECTION 109

Param::Name

The **Name** property is a read/write property that specifies the text used to refer to a parameter. Consult your SAP R/3 documentation to determine the appropriate name of a parameter associated with and Rfc or a BAPI .

Visual Basic Syntax

objParam.Name = strValue
See Also
Param::Field ....
Param::FieldCount........
Param::IsSimple ....
Param::Value................

SECTION 110

Param::Value

The **Value** property can be used to set or retrieve the data that will be passed to a simple parameter. The value set using the Value property should be a simple Variant type such as a float or string. If the parameter is not a simple parameter, the Value property is not supported.

Visual Basic Syntax

objParam.Value = varValue

Error Values

ABERR_PROPERTY_NOT_SUPPORTED	Property is not valid on this instance of the ob

See Also

Param::Field.................
Param::FieldCount........
Param::IsSimple...........
Param::Name................

The **Rfc** object is used to perform a Remote Function Call (RFC) on an SAP R/3 system. The Rfc object allows you to specify the RFC being called by name and to add any desired simple parameters, structure parameters and inbound/outbound tables associated with the RFC.

The Program ID for this object is **abR3Proxy.Rfc**.

Properties	Methods
Rfc::InTables................	Rfc::Execute..................
Rfc::Name....................	
Rfc::OutTables.............	
Rfc::Params..................	
Rfc::Results..................	

SECTION 111

Rfc::Execute

The **Execute** method is used to perform an Rfc call created using the Rfc object. The Execute method takes one parameters. The first parameter is an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to complete the call. When the Execute method is called, the Rfc object attempts to connect to the R/3 system identified by the connection object and to perform the RFC according the current properties of the Rfc object. The Rfc object executes the necessary calls using the SOAP protocol and attempts to connect to a SOAP router where an R/3 proxy object is registered.

Visual Basic Syntax

objRfc.Execute objConnection

Error Values

ABERR_CONNECTION_FAILED	Could not connect to R/3
ABERR_SOAP_CALL_FAILED	The SOAP router returned an error or it is una
ABERR_SOAP_PREPARE_FAILED	Could not create call envelope.
ABERR_NAME_NOTSPECIFIED	Name of RFC is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECI FIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIE D	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIE D	Language specifier not assigned to connection

See Also

Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount.....
Rfc::OutTables.............
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue....................
Rfc::ResultCount..........
Rfc::Results..................

SECTION 112

Rfc::InTables

The **InTables** property can be used to create an inbound table or to retrieve a previously created inbound table. This property takes one string parameter that specifies the name of the table desired. The InTables property returns a reference to a Table object that can be used to add fields and values to the inbound table. If the table, specified in the parameter to the InTables property, does not exist it is created.

Visual Basic Syntax

objTable = objRfc.InTables(strName)

Error Values

ABERR_TABLE_NOTFOUND	Unable to retrieve named table
ABERR_TABLECREATE_FAILED	Could not create table parameter.

See Also

Rfc::Execute.................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 113

Rfc::InTableCount

The **InTableCount** property returns the total number of inbound tables currently associated with a Rfcobject. An inbound table can be added to the Rfc object using the InTables property.

```
Visual Basic Syntax
```

lngValue = objRfc.InTableCount

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 114

Rfc::LoadRequest

The **LoadRequest** method recreates a previously created RFC from a SOAP envelope. The LoadRequest method takes one string parameter that is used to pass the SOAP envelope as text to the method. The Rfc object uses the envelope to build the necessary table, parameter and field objects associated with the call such that they can be accessed using the methods and properties of the object. Typically the AsyncPost object is used along with the LoadRequest method to recreate a RFC that was called asynchronously using the Queue method.

Visual Basic Syntax

objRfc.LoadRequest strValue

Error Values

ABERR_SOAPPROXYCREATE_FAILED	Unable to create SOAP proxy object.
ABERR_PARSEREQUEST_FAILED	Could not interpret envelope.

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 115

Rfc::LoadResponse

The **LoadResponse** method imports the results of a RFC into an Rfc object. The LoadResponse method takes one string parameter that is used to pass a SOAP envelope as text to the method. The Rfc object uses the envelope to build the necessary result parameter and outbound table objects returned by the call such that they can be accessed using the methods and properties of the object. Typically the AsyncPost object is used along with the LoadResponse method to handle a response returned from a RFC that was called asynchronously using the Queue method.

`Visual Basic Syntax`

objRfc.LoadResponse strValue

`Error Values`

ABERR_SOAPPROXYCREATE_FAILED	Unable to create SOAP proxy object.
ABERR_PARSEREQUEST_FAILED	Could not interpret envelope.
ABERR_FUNCTIONEXECUTE_FAILED	R/3 function returned an error.

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 116

Rfc::Name

The **Name** property is a read/write property that holds the text used to refer to the RFC associated with a Rfc object. The name property is a string that must be assigned before the Execute or Queue methods are called. The text assigned to the Name property should be equivalent to the name of a valid RFC active on the destination R/3 system. Consult you SAP system and/or documentation for more information on RFC names.

Visual Basic Syntax

objRfc.Name = strValue

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 117

Rfc::OutTableCount

The **OutTableCount** property returns the total number of outbound tables currently associated with an Rfc object. An outbound table can be retrieved from the Rfc object using the OutTables property.

Visual Basic Syntax

lngValue = objRfc.OutTableCount
See Also
Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTables.............
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue....................
Rfc::ResultCount..........
Rfc::Results..................

SECTION 118

Rfc::OutTables

The **OutTables** property is used to retrieve an outbound table returned by a RFC. This property takes one string parameter that specifies the name of the table desired. The OutTables property returns a reference to a Table object that can be used to access the fields and values contained in the table. If the table, specified in the parameter to the OutTables property, does not exist an error occurs. The Execute or LoadResponse method should be successfully called before attempting to access the outbound tables returned by an RFC.

Visual Basic Syntax

Set objTable = objRfc.OutTables(strName)

Error Values

ABERR_TABLE_NOTFOUND	Unable to retrieve named table

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue....................
Rfc::ResultCount..........
Rfc::Results..................

SECTION 119

Rfc::Params

The **Params** property is used to create or retrieve an argument that will be passed to a RFC. The Params property takes one string parameter that identifies an argument associated with a RFC. If the parameter does not exist it is created when the params property is called. This property returns a reference to a Param object. The methods and properties of the Param object can be used to add values to the RFC parameter.

Visual Basic Syntax

Set objParam = objRfc.Params(strName)

Error Values

ABERR_PARAMETER_NOTFOUND	Parameter could not be retrieved.
ABERR_CREATEPARAMETER_FAILED	Unable to create new parameter.

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount.....
Rfc::OutTables.............
Rfc::ParamCount..........
Rfc:Queue....................
Rfc::ResultCount..........
Rfc::Results..................

SECTION 120

Rfc::ParamCount

The **ParamCount** property returns the total number of parameters currently associated with an Rfc object. A parameter can be added to the Rfc object using the Params property.

```
Visual Basic Syntax
```

lngValue = objRfc.ParamCount

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc:Queue ....
Rfc::ResultCount..........
Rfc::Results..................

SECTION 121

Rfc:Queue

The **Queue** method is used to submit a RFC to the AsyncPost object. When a RFC is submitted to the AsyncPost object it is placed in a queue of asynchronous function calls that will be executed when a Tx-Sync operation is performed by the user. The response returned from the RFC called is stored in the AsyncPost object after the Tx-Sync operation.

The Queue method takes three parameters. The first parameter is an instance of a Connection object created using the abR3Proxy library. The connection object contains the logon and R/3 destination information necessary to complete the call. The second parameter, the *name* parameter, is a string value used to identify the queue of asynchonous calls associated with the application. The third parameter, the *id* parameter, is a string value used to assign an custom identifier to the asynchronous call. The *name* and *id* parameters can be used to retrieve the response to the RFC call from the AsyncPost object after the call has been executed during a Tx-Sync operation.

Visual Basic Syntax

objRfc.Queue objConnection, strName, strId

Error Values

ABERR_NAME_NOTSPECIFIED	Name of the RFC is not set.
ABERR_URL_NOTSPECIFIED	The URL to the SOAP router has not been set.
ABERR_DESTINATION_NOTSPECIFIED	System destination not assigned to connection
ABERR_USER_NOTSPECIFIED	SAP user not assigned to connection object.
ABERR_PASSWORD_NOTSPECIFIED	SAP password not assigned to connection obje
ABERR_CLIENT_NOTSPECIFIED	System client number not assigned to connecti
ABERR_LANGUAGE_NOTSPECIFIED	Language specifier not assigned to connection

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc::ResultCount..........
Rfc::Results..................

SECTION 122

Rfc::ResultCount

The **ResultCount** property returns the total number of return parameters currently associated with an Rfc object. A return parameter can be retrieved from the Rfc object using the Results property.

Visual Basic Syntax

lngValue = objRfc.ResultCount

See Also

Rfc::Execute.................
Rfc::InTables................
Rfc::InTableCount........
Rfc::LoadRequest.........
Rfc::LoadResponse.......
Rfc::Name....................
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params..................
Rfc::ParamCount..........
Rfc:Queue....................
Rfc::Results..................

Section 123

Rfc::Results

The **Results** property is a read-only property that returns a Param object corresponding to an argument returned by a RFC. This property takes one string value corresponding to the name of the return parameter desired. Return parameters are only available after the Execute or LoadResponse methods have been successfully called. Consult your SAP system and/or documentation for the names of available RFCs and associated return parameters.

Visual Basic Syntax

Set objParam = objRfc.Results(strValue)

Error Values

ABERR_PARAMETER_NOTFOUND	Parameter could not be retrieved.

See Also

Rfc::Execute ....
Rfc::InTables ....
Rfc::InTableCount ....
Rfc::LoadRequest ....
Rfc::LoadResponse.......
Rfc::Name ....
Rfc::OutTableCount .....
Rfc::OutTables ....
Rfc::Params ....
Rfc::ParamCount ....
Rfc:Queue ....
Rfc::ResultCount ....

The **Segment** object allows an application to create and edit segments associated with an IDoc object. Each R/3 IDoc can be thought of as a tree-like structure where the branches are IDoc segments. A segment may be attached to the *root* of the IDoc or it may be a *child* or another segment. A segment is created through the AddSegment method of either the IDoc object or an existing Segment object therefore it is typically unnecessary to manually create an instance of a Segment object from the program id.

The Program ID for this object is **AbR3Proxy.Segment**.

Properties	Methods
Segment::Fields	Segment::AddSegment ..
Segment::FieldCount	
Segment::Name	
Segment::SegmentCou nt	
Segment::Segments.......	

SECTION 124

Segment::AddSegment

The **AddSegment** method is used to appends a child segment to a Segment object. The AddSegment method takes one string parameter indicating the segment type name. Consult your system and/or R/3 documentation for more information on available IDocs and their corresponding segments.

Visual Basic Syntax

```
Set objSegment = objSegment.AddSegment( strValue )
```

See Also

Segment::Fields............
Segment::FieldCount....
Segment::Name............
Segment::SegmentCou
nt..................................
Segment::Segments.:.....

SECTION 125

Segment::Fields

The **Fields** property is a read/write property that is used to set and retrieve the values associated with a Segment object. The Fields property takes one parameter that specifies the name or index number of the field. When the name of the desired field is passed to the Fields property as a string, the value of the named field is set or retrieved directly. When the index number of the field is passed to the Fields property, a reference to a Field Object is returned.

To create a new field simply assign the value desired to the Fields property and pass the name of the new field as a string to the parameter.

Visual Basic Syntax

```
objSegment.Fields( strName ) = strValue
Set objField = objSegment.Fields( intIndex )
```

Error Values

ABERR_FIELD_NOT_FOUND	The field specified was not found.

See Also

Segment::AddSegment .
Segment::FieldCount ....
Segment::Name ....
Segment::SegmentCou
nt ....
Segment::Segments.......

SECTION 126

Segment::FieldC_unt

The **FieldCount** property returns the total number of fields currently associated with a structure. A field can be added to a segment using the Fields property.

```
Visual Basic Syntax
```

lngValue = objSegment.FieldCount
See Also
Segment::AddSegment .
Segment::Fields ....
Segment::Name ....
Segment::SegmentCou
nt ....
Segment::Segments.......

SECTION 127

Segment::Name

The **Name** property is a read-only property that returns the segment type assigned to an IDoc segment. The segment type is assigned when the segement is created using the AddSegment method of either the IDoc or Segment ogjects. Consult your R/3 system and/or documentation to determine the appropriate segment type names associated with the available IDocs.

Visual Basic Syntax

strValue = objSegment.Name
See Also
Segment::AddSegment .
Segment::Fields ....
Segment::FieldCount ....
Segment::SegmentCou
nt ....
Segment::Segments.......

SECTION 128

Segment::SegmentCount

The **SegmentCount** property returns the total number of child segments currently associated with a Segment object. A segment can be added to the Segment object using the AddSegment method.

Visual Basic Syntax

lngValue = objSegment.SegmentCount
See Also
Segment::AddSegment .
Segment::Fields ....
Segment::FieldCount ....
Segment::Name ....
Segment::Segments.......

SECTION 129

Segment::Segments

The **Segments** property is a read-only property that returns an instance of a Segment object. The Segments property is used to retrieve a child-segment associated with a Segment object. The Segments property takes one string parameter that is used to indicate the segment type name of the desired child-segment. If the child-segment does not exist an error is raised.

```
Visual Basic Syntax
```

Set objSegment = objSegment.Segments(strValue)

```
Error Values
```

ABERR_SEGMENT_NOT_FOUND	The segment specified was not found.

See Also

Segment::AddSegment .
Segment::Fields ....
Segment::FieldCount ....
Segment::Name ....
Segment::SegmentCou
nt ....

The **Table** object is used to access and edit inbound and outbound table parameters associated with Rfc and BAPI calls.table parameter. A inbound table parameter is created through the InTables property of the Rfc and BAPI objects. Outbound tables are automatically generated when the response from a call is received. Therefore it is typically unnecessary to manually create an instance of a Table object from the program id.

The Table object allows applications to acces the *records* (rows) and *fields* (columns) of a table structure. The Table object has a cursor that is used to navigate between records in a table using the MoveFirst , MoveNext , and MoveLast methods. Once the cursor is positioned on a record, the Fields property can be used to access the values of each column of the table in the record.

The Program ID for this control is **abR3Proxy.Table**.

Properties	Methods
Table::EOF..................	Table::AddRecord.........
Table::FieldCount.........	Table::MoveFirst...........
Table::Fields................	Table::MoveNext..........
Table::Name................	Table::MoveLast...........
Table::RecordCount......	
Table::RecordNumber...	

SECTION 130

Table::AddRecord

The **AddRecord** method adds a new row to an inbound table structure. The table cursor is positioned on the new row after the AddRecord method is called. After the AddRecord method is called the Fields property should be called to add each column of the table to the new row, regardless of whether a value will be placed in the field on the new row. This ensures that the table being created maintains a uniform structure.

If the table represented by the Table object is an outbound table, new rows cannot be added to the table and the AddRecord method will fail if called.

Visual Basic Syntax

```
objTable.AddRecord
```

See Also

Table::EOF...................
Table::FieldCount.........
Table::Fields.................
Table::MoveFirst..........
Table::MoveLast...........
Table::MoveNext..........
Table::Name.................
Table::RecordNumber...
Table::RecordCount......

SECTION 131

Table::EOF

The **EOF** property is a read-only property that returns a boolean value indicating whether the table cursor has moved past the last record. If the value returned is **true**, the table cursor has moved past the last record in the table structure. If the value returned is **false**, the table cursor is positioned on a valid record.

```
Visual Basic Syntax
```

boolValue = objTable.EOF

See Also

Table::AddRecord ....
Table::FieldCount ....
Table::Fields ....
Table::MoveFirst ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

Section 132

Table::FieldCount

The **FieldCount** property returns the total number of columns currently included in a record of a table. A field can be added to a record using the Fields property of the Table object.

Visual Basic Syntax

lngValue = objTable.FieldCount
See Also
Table::AddRecord ....
Table::EOF ....
Table::Fields.................
Table::MoveFirst ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

SECTION 133

Table::Fields

The **Fields** property is a read/write property that is used to create columns and retrieve column values in a record of a table. The Fields property takes one parameter that specifies the name or index number of the field. When the name of the desired field is passed to the Fields property as a string, the value of the named field is set or retrieved directly. When the index number of the field is passed to the Fields property, a reference to a Field Object is returned.

To create a new column, pass the name of the column to the parameter and assign the value of the new column to the Field property.

Visual Basic Syntax

```
objTable.Fields( strName ) = strValue
Set objField = objTable.Fields( intIndex )
```

Error Values

ABERR_FIELD_NOT_FOUND	The field specified was not found.

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount ....
Table::MoveFirst ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

Section 134

Table::MoveFirst

The **MoveFirst** method positions the cursor on the first record in a table structure. If the table is empty an error is raised. When a Table object is first retrieved, the table cursor is positioned just before the first record of the table structure. After a Table oject is first retrieved, the MoveFirst or MoveNext method should be called before attempting to access the columns of a valid record.

Visual Basic Syntax

objTable.MoveFirst

Error Values

ABERR_INVALID_RECORD_NUMBER	Cannot move to the specified record.

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount ....
Table::Fields ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

SECTION 135

Table::MoveLast

The **MoveLast** method positions the table cursor on the last record in the table. If the table is empty an error is raised.

Visual Basic Syntax

objTable.MoveLast

Error Values

ABERR_INVALID_RECORD_NUMBER	Cannot move to the specified record.

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount ....
Table::Fields ....
Table::MoveFirst ....
Table::MoveNext ....
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

SECTION 136

Table::MoveNext

The **MoveNext** method moves the current cursor position up by one record. If the table is empty or the cursor is positioned on the last record an error is raised. After the cursor has been postioned on the specified record the Fields property can be used to access the column values of the record.

Visual Basic Syntax

objTable.MoveNext

Error Values

ABERR_INVALID_RECORD_NUMBER	Cannot move to the specified record.

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount ....
Table::Fields ....
Table::MoveFirst ....
Table::MoveLast...........
Table::Name ....
Table::RecordNumber...
Table::RecordCount......

SECTION 137

Table::Name

The **Name** property is a read/write property that holds the text used to refer to the table. When an inbound table is created through the Rfc or BAPI object, the name of the new table is specified as a parameter to the InTables property. For an inbound table, the name property can be used to modify or view the name after the table is created. The text assigned to the Name property should be equivalent to the name of a valid table parameter associated with a BAPI or RFC on the destination R/3 system. Consult you SAP system and/or documentation for more information on table parameter names.

When an outbound table is retrieved after the response to a BAPI or RFC call has been received, the name property is read-only and cannot be used to modify the value.

Visual Basic Syntax

objTable.Name = strValue
See Also
Table::AddRecord ....
Table::EOF ....
Table::FieldCount.........
Table::Fields.................
Table::MoveFirst..........
Table::MoveLast...........
Table::MoveNext..........
Table::RecordNumber...
Table::RecordCount......

SECTION 138

Table::RecordNumber

The **RecordNumber** property returns the index of the record where the table cursor is currently positioned. The index of the first record in a table structure is zero (0).

```
Visual Basic Syntax
```

lngValue = objTable.RecordNumber

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount.........
Table::Fields ....
Table::MoveFirst ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name ....
Table::RecordCount......

SECTION 139

Table::RecordCount

The **RecordCount** property returns the total number of records currently included in a table. A record can be added to a table using the AddRecord method of the Table object.

```
Visual Basic Syntax
```

lngValue = objTable.RecordCount

See Also

Table::AddRecord ....
Table::EOF ....
Table::FieldCount.........
Table::Fields ....
Table::MoveFirst ....
Table::MoveLast...........
Table::MoveNext..........
Table::Name.................
Table::RecordNumber...

CHAPTER 9

abRas (Remote Access Service)

The **abras** library provides simplified tools for accessing the Remote Access Service (RAS) APIs on Windows Powered devices. With access to the RAS APIs applications can establish connections with remote hosts over a variety of transport mechanisms as well as check to see if a particular connection is active.

The **RasConn** object provides access to a named Remote Access Service phonebook entry. Each entry specifies a predefined transport mechanism (such as modem, USB, etc...) and a connection configuration (baudrate, phone number, etc...). The RasConn object allows applications to utilize existing phonebook entries. The creation, configuration, and deletion of phonebook entries is an administrative duty which is centrally managed through the system Console.

The Program ID for this control is **abras.RasConn**.

Methods

SECTION 140

RasConn::Dial

The **Dial** method opens a Remote Access Service (RAS) connection. This method takes one string parameter indicating the unique phonebook name of the connection entry. When this method is called a status window typically appears displaying the progress of the dialing operation. If the connection entry has been configured to prompt the user for a username/password combination before opening a connection, this prompt will appear before the progress window.

The status of a connection can be determined using the IsConnected method of the RasConn object.

`Visual Basic Syntax`

objRasConn.Dial strName

`Error Values`

ABERR_ENTRYNOTFOUND	Entry specified is not found or is not ava
ABERR_RASOPENFAILED	Unable to open phonebook entry.
ABERR_RASDIALFAILED	Unable to establish connection.
ABERR_RASTIMEOUT	Connection timed out while opening.

See Also

RasConn::HangUp........
RasConn::IsConnected..

SECTION 141

RasConn::HangUp

The **HangUp** method disconnects an open Remote Access Service (RAS) connection. The method takes one string parameter indicating the unique phonebook name of the connection. This method has no effect if the connection specified is not currently open.

Visual Basic Syntax

objRasConn.HangUp strName

See Also

SECTION 142

RasConn::IsConnected

The **IsConnected** method is used to check the status of a RAS connection entry. If the connection is open, the IsConnected method returns true. If the connection is closed, the method returns false. The IsConnected method takes one string parameter indicating the unique name of the connection entry.

Visual Basic Syntax

boolValue = objRasConn.IsConnected(strName)

Error Values

ABERR_ENTRYNOTFOUND	Entry specified is not found or unavailabl

See Also

RasConn::Dial..............
RasConn::HangUp........

The **RasEnum** control provides a listing of all connections available on a device's Remote Access Service (RAS) phonebook. Each connection entry is identified by an alphanumeric name which can be used along with the RasConn object to make use of the connection.

The Program ID for this control is **abras.RasEnum**.

Properties

RasEnum::Count...........
RasEnum::Item.............

Section 143

RasEnum::Count

The **Count** property returns the total number of connection entries listed in the phonebook.

`Visual Basic Syntax`

lngCount = objRasEnum.Count
See Also
RasEnum::Item.............

SECTION 144

RasEnum::Item

The **Item** method returns a string value idicating the name of a connection entry. The Item method takes one parameter which is a long value specifying the list postion of the entry desired. The index of the first entry in the list is zero (0).

```
Visual Basic Syntax
```

strValue = objRasEnum.Item(lngIndex)

See Also

RasEnum::Count...........

CHAPTER 10

abShell (Shell Control)

The **abShell** library allows applications to take full control of the interaction between a user and a device. Most Windowsã Powered devices have hardware buttons, operating system menus and icons which are always available to the user, allowing him or her freedom to exit and re-enter an application using a variety of methods. The tools provided by the **abshell** library enable an application to take back some of this freedom from a potential user with the intent of keeping them focused on a particular task or sequence of steps.

The **Shell** object allows applications to interact with the core user interface elements of the device operating system. The Shell object allows software user interface elements to be hidden and shown and hardware user interface elements to be enabled and disabled such that an application can take full control of the mechanisms available to the user.

The Program ID for this object is **abshell.Shell**.

Properties	Methods
Shell::LockAppKey	Shell::ShowSipButton ...
	Shell::ShowStartIcon.....
	Shell::ShowTaskBar......

SECTION 145

Shell::LockAppKey

The **LockAppKey** property sets the operating state of the application launch buttons of a device. When this property is set to true, the application launch buttons are activated and the user is able to use them to directly launch applications. When this property is false the buttons are deactivated and will have no effect when pressed.

Application launch buttons typically start standard applications which are always present on devices such as Calendar applications and Address Book applications. These buttons can also typically be configured to launch custom applications when present on a device. Consult the operating system documentation for more info on this topic.

```
Visual Basic Syntax
```

objShell.LockAppKey = boolValue

See Also

Shell::ShowSipButton...
Shell::ShowStartIcon ....
Shell::ShowTaskBar .....

SECTION 146

Shell::ShowSipButton

The **ShowSipButton** method is used to control the Supplementary Input Panel (*SIP*) button. The SIP button is used to activate the operating system's virtual keypads which can be used to enter text using a variety of methods.

This method takes one boolean value which indicates the desired state of the SIP button. When this method is called with a *false* value, the SIP button is hidden and will not be accesible to the user while your application is active. When the method is called with a *true* value, the SIP button is shown and the user will have access to the virtual keypads. This method is particularly useful when forcing a user to use a specific method of data input such as a scanner.

Visual Basic Syntax

objShell.ShowSipButton boolValue

Error Values

ABERR_SHFULLSCREEN	Error executing SHFullScreen.

See Also

Shell::LockAppKey ......
Shell::ShowStartIcon ....
Shell::ShowTaskBar .....

SECTION 147

Shell::ShowStartIcon

The **ShowStartIcon** method controls the Start Icon which provides access to the main operating system menu. The Start Icon (located in top left corner of Pocket PC devices) lists the majority of the applications installed on a device and provices access to a variety of operating system tools for configuring and customizing a device.

The ShowStartIcon method takes two parmeters: the first is a boolean value indicating the desired state of the Start icon; the second is an optional long value designating the handle of the application's main window. When this method is called with a *false* value, the Start Icon is hidden and the main operating system menu becomes unavailable. When this method is called with a *true* value, the Start icon is shown and the main operating system menu is available from within the application.

Visual Basic Syntax

objShell.ShowStartIcon boolValue, [hWnd]

Return Values

ABERR_SHFULLSCREEN	Error executing SHFullScreen.

See Also

Shell::LockAppKey ......
Shell::ShowSipButton...
Shell::ShowTaskBar .....

SECTION 148

Shell::ShowTaskBar

The **ShowTaskBar** method controls the operating system Task Bar. On Pocket PC devices, the Task Bar appears as a fixed height strip on the bottom of the device screen. This strip contains icons which may launch application specific functions or general operating system functions.

The ShowTaskBar method takes two parameters: the first is a boolean value indicating the desired state of the Task Bar; the second is an optional long value designating the handle of the application's main window. When the ShowTaskBar method is called with a *false* value, the Task Bar is hidden and the window is enlarged to cover the space vacated by the Task Bar. When the method is called with a *true* value, the Task Bar is shown along with any icons that have been placed there by applications and/or the operating system.

Visual Basic Syntax

objShell.ShowTaskBar boolValue, [hWnd]

Error Values

ABERR_SHFULLSCREEN	Error executing SHFullScreen.

See Also

Shell::LockAppKey......
Shell::ShowSipButton...
Shell::ShowStartIcon....

The **Browser** object enables applications to modify the user interface of the Pocket Internet Explorer. The Pocket Internet Explorer is the default browser provided by devices operating on the Pocket PC platform and is typically used as the primary tool to interact with HTML and ASP based applications.

The Program ID for this control is **abshell.Browser**.

Methods

Browser::ShowAddres
sBar............................
Browser::ShowMenuB
ar................................

Section 149

Browser::ShowAddressBar

The **ShowAddressBar** method controls the Address Bar of the Pocket Internet Explorer. The Address Bar is a combo box displayed near the top of the Pocket Internet Explorer which displays the URL of the currently displayed page and allows the URL to be directly modified in order to navigate to a new location.

The ShowAddressBar method takes no parameters. When the method is called the state of the Address Bar is toggled. If the Address Bar is visible, a call to this method hides the Address Bar. If it is hidden, a call to this method shows the Address Bar.

`Visual Basic Syntax`

objBrowser.ShowAddressBar

`Return Values`

ABERR_IENOTFOUND	Unable to find IExplorer.

See Also

Browser::ShowMenuB
ar.................................

SECTION 150

Browser::ShowMenuBar

The **ShowMenuBar** method controls the Pocket Internet Explorer's Menu Bar. The Menu Bar appears in the area where the operating system's Task Bar is normally shown and provides acces to various features and options of the Pocket Internet Explorer.

The ShowMenuBar method takes one boolean parameter which is used to indicate the desired state of the explorer's Menu Bar. When this a *false* value is specified the Menu Bar is hidden; when this parameter is a *true* value the Menu Bar is shown.

`Visual Basic Syntax`

objBrowser.ShowMenuBar boolValue

`Error Values`

ABERR_IENOTFOUND	Unable to find IExplorer.

See Also

Browser::ShowAddres
sBar..............................

CHAPTER 11

abSoap (SOAP)

The **absoap** library enables applications to consume web services and is thus a key element of the Device Framework. This library provides all of the tools necessary to make use of the Simple Object Access Protocol (SOAP). SOAP allows applications to communicate with remote resources using the HTTP protocol. SOAP's XML based message format, known as an *envelope*, allows applications to communicate regardless of computing platform or programming language. SOAP envelopes can be easily created, transmitted and received using the **absoap** library.

The **SoapProxy** object is the central tool used to process SOAP messages. The SoapProxy object allows applications to create, send and receive SOAP envelopes as well as perform a number of helpful additional tasks.

SOAP Routers

The SoapProxy object sends messages to and receives messages from a SOAP router. A SOAP router is software residing on a web server that receives and processes SOAP messages.

The Program ID for this control is **absoap.SoapProxy**.

Properties	Methods
SoapProxy::ClientCertificate	SoapProxy::Execute ......
SoapProxy::CoerceResults	SoapProxy::GetFile ....
SoapProxy::FaultActor .	SoapProxy::PutFile........
SoapProxy::FaultCode ..	
SoapProxy::FaultDetail..................................	
SoapProxy::FaultString..................................	
SoapProxy::MethodName	
SoapProxy::ObjectName	
SoapProxy::RequestText.................................	
SoapProxy::RequestXML	
SoapProxy::ResponseText.............................	
SoapProxy::ResponseXML	
SoapProxy::RouterURL..................................	
SoapProxy::SoapFaultExists...........................	
SoapProxy::UseXoomSecurity	

SECTION 151

SoapProxy::ClientCertificate

The **ClientCertificate** property allows you to read and write a digital client certificate to the SoapProxy object. Digital Client Certificates are required to access a systemserver when the security options are enabled on the server. When these security options are enabled, asystem server will only accept client certificates that have been issued through it's iternal certificate authority. The SoapProxy object automatically loads this digital certificate when it is instantiated. As a result It is generally not necessary to manually set or read this property.

Visual Basic Syntax

strValue = objSoapProxy.ClientCertificate

See Also

SoapProxy::CoerceRes
ults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetai
l...................................
SoapProxy::FaultStrin
g..................................
SoapProxy::GetFile.......
SoapProxy::MethodNa
me ....
SoapProxy::ObjectNa
me ....
SoapProxy::PutFile ....
SoapProxy::RequestTe
xt.................................
SoapProxy::RequestX
ML ....
SoapProxy::Response
Text ....
SoapProxy::Response
XML ....
SoapProxy::Result ....
SoapProxy::RouterUR
L..................................
SoapProxy::SoapFault
Exists............................
SoapProxy::UseXoom
Security ....

SECTION 152

SoapProxy::CoerceResults

The **CoerceResults** property determines whether the SoapProxy object should attempt to convert return values to an appropriate Variant datatype. If this property is set to false, the default, all return values are returned as strings. When set to true, the SoapProxy object will attempt to convert any return values from a SOAP call to a datatype which matches the value when the value is requested using the SoapProxy Result property . For instance, a numeric result (ex: "354") would be converted to an integer; a boolean keyword result (ex: "false") would be converted to a boolean.

`Visual Basic Syntax`

objSoapProxy.CoerceResults = boolValue
See Also
SoapProxy::ClientCert
ificate ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode..
SoapProxy::FaultDetai
l....................................
SoapProxy::FaultStrin
g..................................
SoapProxy::GetFile.......
SoapProxy::MethodNa
me ....
SoapProxy::ObjectNa
me ....
SoapProxy::PutFile.......
SoapProxy::RequestTe
xt..................................
SoapProxy::RequestX
ML ....
SoapProxy::Response
Text..............................
SoapProxy::Response
XML ....
SoapProxy::Result ....
SoapProxy::RouterUR
L...................................
SoapProxy::SoapFault
Exists............................
SoapProxy::UseXoom
Security ....

SECTION 153

SoapProxy::Execute

The **Execute** method sends the current SOAP request stored in the SoapProxy object to the address specified in the RouterURL property. The Execute method is a synchronous method and it will not return until it receives a response from the specified server or until the request times out. The MethodName property and ObjectName property must be set before the Execute method is called.

This method returns a long value indicating the HTTP status code of the request. Note that the value returned here is not a response to the SOAP envelope itself but instead is a code returned in response to the HTTP request which contained the SOAP envelope. If this code indicates success, the response to the SOAP envelope can be obtained using SoapProxy Result property and SoapFaultExists property .

Visual Basic Syntax

intValue = objSoapProxy.Execute

Error Values

ABERR_NOROUTERURL	RouterURL property has not been set
ABERR_NOMETHOD	MethodName property has not been set
ABERR_NOOBJECT	ObjectName property has not been set

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail ....
SoapProxy::FaultString ....
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText ....
SoapProxy::RequestXML ....

SECTION 154

SoapProxy::FaultActor

The **FaultActor** property returns a string value indicating the entity that raised the error which caused a SOAP fault. A SOAP fault is the type of SOAP envelope returned when errors occur during the execution of the SOAP request on the server. The exact contents of this property are dependent on the type and configuration of the SOAP router used by the server and possibly the programming language of the object being called.

```
Visual Basic Syntax
```

strValue = objSoapProxy.FaultActor

See Also

SoapProxy::ClientCert
ificate ....
SoapProxy::CoerceRes
ults ....
SoapProxy::Execute......
SoapProxy::FaultCode..
SoapProxy::FaultDetai
l...................................
SoapProxy::FaultStrin
g..................................
SoapProxy::GetFile.......
SoapProxy::MethodNa
me ....
SoapProxy::ObjectNa
me ....
SoapProxy::PutFile.......
SoapProxy::RequestTe
xt.................................
SoapProxy::RequestX
ML ....
SoapProxy::Response
Text.............................
SoapProxy::Response
XML ....
SoapProxy::Result ....
SoapProxy::RouterUR
L..................................
SoapProxy::SoapFault
Exists............................
SoapProxy::UseXoom
Security ....

Section 155

SoapProxy::FaultCode

The **FaultCode** property returns a string value indicating an error code for the error which caused a SOAP fault. A SOAP fault is the type of SOAP envelope returned when errors occur during the execution of the SOAP request on the server. The exact contents of this property are dependent on the type and configuration of the SOAP router used by the server and possibly the programming language of the object being called.

Visual Basic Syntax

strValue = objSoapProxy.FaultCode

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultDetail..................................
SoapProxy::FaultString..................................
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText.................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

Section 156

SoapProxy::FaultDetail

The **FaultDetail** property returns a string value that is a verbose description of the error which caused a SOAP fault. A SOAP fault is the type of SOAP envelope returned when errors occur during the execution of the SOAP request on the server. The exact contents of this property are dependent on the type and configuration of the SOAP router used by the server and possibly the programming language of the object being called.

`Visual Basic Syntax`

strValue = objSoapProxy.FaultDetail

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode..
SoapProxy::FaultString ....
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText ....
SoapProxy::RequestXML ....
SoapProxy::ResponseText ....
SoapProxy::ResponseXML ....
SoapProxy::Result........
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists............................
SoapProxy::UseXoomSecurity ....

SECTION 157

SoapProxy::FaultString

The **FaultString** property returns a string value that is a brief description of the error which caused a SOAP fault. A SOAP fault is the type of SOAP envelope returned when errors occur during the execution of the SOAP request on the server. The exact contents of this property are dependent on the type and configuration of the SOAP router used by the server and possibly the programming language of the object being called.

Visual Basic Syntax

strValue = objSoapProxy.FaultString

See Also

SECTION 158

SoapProxy::GetFile

The **GetFile** method sends an HTTP GET request to the URL specified in the RouterURL property of the SoapProxy object and writes the file data returned to a local file. This method takes two parameters: first the local path and filename and second a boolean indicating whether this file shoud be overwritten if it exists. The GetFile method will implicitly use the default XOOM digital certificate to authorize the request if the UseXoomSecurity property is enabled. The GetFile method is useful for retrieving large binary files from remote locations.

The return value of this method is the HTTP status code indicating the outcome of the GET request.

```
Visual Basic Syntax
intValue = objSoapProxy.GetFile( strFilename,
bOverwrite)
Error Values
```

ABERR_HEADERS	A connection with the server could not be establis
ABERR_LOCALFILE	The local file could not be created while performi
ABERR_REMOTEFILEREAD	Error reading remote file
ABERR_LOCALFILEWRITE	Error writing to local file

See Also

SoapProxy::ClientCert
ificate ....
SoapProxy::CoerceRes
ults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode..
SoapProxy::FaultDetai
l...................................
SoapProxy::FaultStrin
g...................................
SoapProxy::MethodNa
me ....
SoapProxy::ObjectNa
me ....
SoapProxy::PutFile.......
SoapProxy::RequestTe
xt.................................
SoapProxy::RequestX
ML ....
SoapProxy::Response
Text.............................
SoapProxy::Response
XML ....
SoapProxy::Result........
SoapProxy::RouterUR
L...................................
SoapProxy::SoapFault
Exists............................
SoapProxy::UseXoom
Security ....

SECTION 159

SoapProxy::MethodName

MethodName is a string property that sets the name of the remote object's member function that will be called by the pending request. This name is typically case sensitive if the remote object is written in a case sensitive programming language such as Java.

When the MethodName property is set, the SoapProxy object's Parameters collection is initialized. Any parameters that may have been added to the SoapProxy object prior to setting the MethodName property will be discarded.

```
Visual Basic Syntax
objSoapProxy.MethodName = strValue
Error Values
```

ABERR_INVALIDDOCUMENT	Invalid internal MSXML document.

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail...................................
SoapProxy::FaultString...................................
SoapProxy::GetFile.......
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText.................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

SECTION 160

SoapProxy::ObjectName

The **ObjectName** property sets the Uniform Resource Identifier (URI) of the remote object that will be called in the pending SOAP request. The URI is essentially the name under which a web service is listed. Typically this name, or a portion of it, is derived from the actual class/object name that is called by the SOAP router to service a request. The ObjectName property must be set before the Execute method of the SoapProxy object can be called.

Visual Basic Syntax

```
objSoapProxy.ObjectName = strValue
```

Error Values

ABERR_INVALIDDOCUMENT	Invalid internal MSXML document.

See Also
SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail...................................
SoapProxy::FaultString.................................
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::PutFile.......
SoapProxy::RequestText.................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

Section 161

SoapProxy::PutFile

The **PutFile** method sends an HTTP PUT request to the URL specified in the RouterURL property of the SoapProxy object. The PUT request is used to send files from the device to the server located at the specified URL. This method takes two parameters: first the path and name of the file on the device that will be transferred and second the suggested name for the file on the remote server. The PutFile method will implicitly use the default XOOM digital certificate to authorize the request if the UseXoomSecurity property is enabled.

THis method returns and integer status code indicating the result of the HTTP PUT request.

Visual Basic Syntax

```
intValue = objSoapProxy.PutFile( strPath,
strRemoteName)
```

Error Values

ABERR_INVALIDDOCUMENT	Invalid internal MSXML document.
ABERR_UNSUPPROTO	Unsupported protocol
ABERR_NOCONNECTION	Could not open session to remote host
ABERR_REQUEST	Could not open request for PUT operatioı
ABERR_FILENOTFOUND	The file specified was not found
ABERR_HEADERS	A connection with the server could not bε
ABERR_READ	Unknown error while reading file
ABERR_WRITEREMOTE	Error while writing file to remote server
ABERR_CONNECTIONRESET	Connection reset by remote server

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode..
SoapProxy::FaultDetail...................................
SoapProxy::FaultString.................................
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::RequestText.................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL.................................
SoapProxy::SoapFaultExists............................

SoapProxy::UseXoom
Security ....

SECTION 162

SoapProxy::RequestText

The **RequestText** property is a read-only property which returns the current SOAP request envelope as a text string. The RequestText property is provided for informational purposes and it can be read at any point during the creation or after the execution of a SOAP request without any adverse affects. If no properties have been set for a SOAP request, the RequestText property will return a SOAP envelope template.

`Visual Basic Syntax`

```
strValue = objSoapProxy.RequestText
```

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail ....
SoapProxy::FaultString ....
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestXML ....
SoapProxy::ResponseText............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

Section 163

SoapProxy::RequestXML

The **RequestXML** property is a read-only property which returns the current SOAP request envelope as an XML DOM Document object. The RequestXML property is provided for informational purposes and it can be read at any point during the creation or after the execution of a SOAP request without any adverse affects. If no properties have been set for a SOAP request, the RequestXML property will return a DOM object representing a template SOAP envelope. Refer to the Microsoft XML documentation for more information about the XMLDOMDocument object and the Microsoft DOM object model.

`Visual Basic Syntax`

```
Set objDOMDocument = objSoapProxy.RequestXML
```

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail..................................
SoapProxy::FaultString ....
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText.................................
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

SECTION 164

SoapProxy::ResponseText

The **ResponseText** property is a read-only property which returns the raw text received by the HTTP layer of the SoapProxy object in response to a SOAP request. This property is only available after the Execute method has been called. If an error occurs before a SOAP request reaches it's destination SOAP router, this property will not contain a properly constructed SOAP envelope but instead may contain an HTML document or nothing at all.

Visual Basic Syntax

strValue = objSoapProxy.ResponseText

See Also

Section 165

SoapProxy::ResponseXML

The **ResponseXML** property is a read-only property which returns an XMLDOMDocument representation of the envelope received in response to a SOAP request. This property is only available after the Execute method has been called. If an error occurs before a SOAP request reaches it's destination SOAP router, this property will not contain a valid DOMDocument instance.

Visual Basic Syntax

Set objDOMDocument = objSoapProxy.ResponseXML

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail...................................
SoapProxy::FaultString..................................
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText.................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................
SoapProxy::UseXoomSecurity ....

SECTION 166

SoapProxy::Result

The **Result** property is a read-only property that provides access to the value returned from the method that was called using SOAP. By default, this property returns a string value. However, if the CoerceResults property is enabled this property will return a value with a datatype closest to the actual datatype returned by the remote function. If any type of error occurs during execution of a SOAP request, the Result property will return null.

Visual Basic Syntax

strValue = objSoapProxy.Result

See Also

Section 167

SoapProxy::RouterURL

The **RouterURL** property sets the Uniform Resource Locator (URL) for the SOAP router to be used for a pending SOAP request. The RouterURL property defines the network location to which the SOAP request will be sent when the Execute method is called. The default value of this property is set to the XOOM system router which is automatically stored and updated on the device.

```
Visual Basic Syntax
```

```
objSoapProxy.RouterURL = strValue


'Call the "WebCalculator" object which exposes an
'"Add" function that takes two integers as
'parameters and returns their sum
Public Function SoapAdd(x, y)
Dim objProxy As Object
Dim intSum As Integer

 Set objProxy = CreateObject("absoap.soapproxy")

' set the URL of the router where the calculator object is located
 objProxy.RouterURL = "http://lab.computer1.com:8080/servlet/rpcrouter"
 objProxy.ObjectName = "WebCalculator"
 objProxy.MethodName = "Add"
 objProxy.Parameters.Add "x", x, "integer"
 objProxy.Parameters.Add "y", y, "integer"
 objProxy.CoerceResults = true
 objProxy.Execute

' retrieve the sum of the numbers from the WebCalculator object
 intSum = objProxy.Result

 Set objProxy = Nothing

' return the sum of the numbers
 SoapAdd = intSum

End Function
```

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetail ....
SoapProxy::FaultString ....
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......

SECTION 168

SoapProxy::SoapFaultExists

The **SoapFaultExists** property is a read-only property that is used to determine when an error has been returned from the destination router. A SOAP fault is a specific type of envelope that describes an error that was raised during execution of a SOAP request. If a SOAP fault is detected, a number of properties such as the FaultCode property and FaultString property can be used to retrieve further detail on the underlying error.

Note that errors in the communication layer are not reflected by the SoapFaultExists property. Errors in the communication layer typically occur before a response envelope is returned and can typically be detected by checking the status code returned by the Execute method .

```
Visual Basic Syntax
```

```
boolValue = objSoapProxy.SoapFaultExists


Dim objProxy As Object

 Set objProxy = CreateObject("absoap.soapproxy")

' set router URL, method, object and parameter
' properties....

' execute a soap request
 objProxy.Execute

' check for soap faults
 If objProxy.SoapFaultExists Then
    MsgBox "Warning error: " & objProxy.FaultCode & " - " & _
     objProxy.FaultString
 Else
    MsgBox "Success! Return value is " & objProxy.Result
 End If

 Set objProxy = Nothing
```

See Also

SoapProxy::ClientCert
ificate ....
SoapProxy::CoerceRes
ults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode ..
SoapProxy::FaultDetai
l...................................
SoapProxy::FaultStrin
g ....
SoapProxy::GetFile.......
SoapProxy::MethodNa
me ....
SoapProxy::ObjectNa
me ....
SoapProxy::PutFile ....
SoapProxy::RequestTe
xt ....
SoapProxy::RequestX
ML ....
SoapProxy::Response
Text ....
SoapProxy::Response

XML ....
SoapProxy::Result ....
SoapProxy::RouterUR
L....................................
SoapProxy::UseXoom
Security ....

SECTION 169

SoapProxy::UseXoomSecurity

The **UseXoomSecurity** property is used to enable and disable transmission of a digital client certificate. When enabled, the SoapProxy object uses a digital client certificate to authorize communication with the destination server specified in the RouterURL property . The digital client certificate used is automatically installed on the device by the XOOM system and is automatically loaded by the SoapProxy object. If this client certificate is not present, an attempt to enable this property will result in an error.

Visual Basic Syntax

objSoapProxy.UseXoomSecurity = boolValue

See Also

SoapProxy::ClientCertificate ....
SoapProxy::CoerceResults ....
SoapProxy::Execute......
SoapProxy::FaultActor .
SoapProxy::FaultCode..
SoapProxy::FaultDetail..................................
SoapProxy::FaultString..................................
SoapProxy::GetFile.......
SoapProxy::MethodName ....
SoapProxy::ObjectName ....
SoapProxy::PutFile.......
SoapProxy::RequestText................................
SoapProxy::RequestXML ....
SoapProxy::ResponseText.............................
SoapProxy::ResponseXML ....
SoapProxy::Result ....
SoapProxy::RouterURL..................................
SoapProxy::SoapFaultExists...........................

The **Headers** object is a collection that provides access to the HTTP layer of the absoap control. This object stores HTTP headers that will be added to each HTTP request generated by the SoapProxy object . The methods of the SoapProxy object that generate HTTP requests are the Execute method , the GetFile method , and the PutFile method . The SoapProxy object internally adds specific headers to each request depending on the type of request being made. The Headers object does not provide access to these internal headers but instead allows additional headers to be added to each request.

Properties	Methods
Headers::Item	Headers::Add ....
Headers::Count	Headers::Clear...............
	Headers::Remove ....

SECTION 170

Headers::Add

The **Add** method inserts a new HTTP header into the collection. The Add method takes one parameter that is an instance of an HttpHeader object.

```
Visual Basic Syntax
objHeaders.Add objNewHeader
```

See Also

Headers::Clear..............
Headers::Count.............
Headers::Item ....
Headers::Remove..........

SECTION 171

Headers::Clear

The **Clear** method removes all existing HTTP headers from the collection.

Visual Basic Syntax

```
objHeaders.Clear
```

See Also

Headers::Add................
Headers::Count.............
Headers::Item ....
Headers::Remove..........

SECTION 172

Headers::Count

The **Count** property is a read-only property that returns the total number of HTTP Headers currently in the collection.

`Visual Basic Syntax`

objHeaders`.Add objNewHeader`
See Also
Headers::Add................
Headers::Clear..............
Headers::Item ....
Headers::Remove..........

SECTION 173

Headers::Item

The **Item** method is used to retrieve an HTTP Header from the collection. The Item method takes one parameter that can be an integer or a string, indicating the position or name of the header to be returned. The first header in the collection has an index of 1.

`Visual Basic Syntax`

Set objHeader = objHeaders.`Item(intIndex)`
Set objHeader = objHeaders.`Item(strName)`
See Also
Headers::Add................
Headers::Clear..............
Headers::Count.............
Headers::Remove..........

Section 174

Headers::Remove

The **Remove** method removes an HTTP Header from the collection. The Remove method takes one parameter which is and integer or string indicating the index or name of the header to be removed.

Visual Basic Syntax

```
objHeaders.Remove(intIndex)
objHeaders.Remove(strName)
```

See Also

Headers::Add................
Headers::Clear..............
Headers::Count.............
Headers::Item ....

The **HttpHeader** object is used to represent an HTTP Header in the Headers collection. The HttpHeader object consists of a name-value pair which is set using the properties of the object.

Properties

Header::Name...............
Header::Value...............

Section 175

Header::Name

The **Name** property sets or retrieves the name of an HTTP Header.

```
Visual Basic Syntax
```

objHttpHeader.Name = strName

See Also

Header::Value...............

SECTION 176

Header::Value

The **Value** property sets or retrieves the value associated with an HTTP header.

```
Visual Basic Syntax
```

objHttpHeader.Value = strValue
See Also
Header::Name...............

The **Parameter** object is used to represent an SOAP envelope parameter in the Parameters collection . The Parameter object consists of a name, value and type, each of which is set using the properties of the object.

Properties

Parameter::Name ....
Parameter::Value ....
Parameter::Type ....

SECTION 177

Parameter::Name

The **Name** property sets or retrieves the name of a parameter to a method.

`Visual Basic Syntax`

strValue = objParameter.Name
See Also
Parameter::Value ....
Parameter::Type ....

SECTION 178

Parameter::Value

The **Value** property sets or retrieves the data that will be passed to a parameter of a method.

`Visual Basic Syntax`

strValue = objParameter.Value
See Also
Parameter::Name ....
Parameter::Type ....

Section 179

Parameter::Type

The **Type** property sets or retrieves the datatype of a parameter. The datatype may or may not be required depending on the type of SOAP router used on the destination resource.

```
Visual Basic Syntax
```

strValue = objParameter.Type
See Also
Parameter::Name ....
Parameter::Value ....

The **Parameters** object is a collection of the parameters which will be passed to a method when it is called using the Execute method of the SoapProxy object . The Parameters collection is automatically cleared when the MethodName property of the SoapProxy object is set thus it is necessary to set the MethodName property before manipulating the Parameters collection.

Properties	Method
Parameters::Item	Parameters::Add ....
Parameters::Count	Parameters::Clear ....
	Parameters::Remove......

SECTION 180

Parameters::Add

The **Add** method inserts a new parameter into the Parameters collection. The add method takes three string parameters: Name, Value and Type. **Name** is the variable name used to refer to the parameter. **Value** is the data that will be passed to the parameter when the method is called. **Type** sets the datatype of the parameter. Take the following Java method declaration as an example:

```
public int add(int x, int y) {
...
}
```

In the declaration above, the *add* method takes two parameters. For the first parameter, the name is *x* and the type is *integer*. Likewise the second parameter's name is *y* and its type is *integer*.

Some SOAP routers do not require the type of the variable to be specified.

```
Visual Basic Syntax
objParameters.Add strName, strValue, strType
```

See Also

Parameters::Clear..........
Parameters::Count ....
Parameters::Item...........
Parameters::Remove.....

SECTION 181

Parameters::Clear

The **Clear** method removes all parameters from the collection.

Visual Basic Syntax

```
objParameters.Clear
```

See Also

Parameters::Add ....
Parameters::Count ....
Parameters::Item...........
Parameters::Remove .....

SECTION 182

Parameters::Count

The **Count** property is a read-only property that returns the total number of parameters currently held in the collection.

```
Visual Basic Syntax
```

lngValue = objParameters.Count

See Also

Parameters::Add ....

Parameters::Clear..........

Parameters::Item ....

Parameters::Remove .....

Section 183

Parameters::Item

The **Item** method is used to retrieve a parameter from the collection. The method takes one paramter which can be an integer indicating the index of the item to be retrieved or a string indicating the name of the item to be retrieved. The first parameter added to the collection has an index equal to 1.

Visual Basic Syntax

```
Set objParamater = objParameters.Item(intIndex)
Set objParamater = objParameters.Item(strName)
```

See Also

Parameters::Add ....
Parameters::Clear..........
Parameters::Count ....
Parameters::Remove.....

SECTION 184

Parameters::Remove

The **Remove** method deletes a parameter from the collection. The method takes one paramter which can be an integer indicating the index of the item to be removed or a string indicating the name of the item to be removed. The first parameter added to the collection has an index equal to 1.

```
Visual Basic Syntax
```

objParameters.Remove(intIndex)
objParameters.Remove(strName)

See Also

Parameters::Add ....
Parameters::Clear..........
Parameters::Count ....
Parameters::Item...........

CHAPTER 12

abSoapPing (Ping)

The **abSoapPing** library provides tools that allow applications to determine the current state of a device's network connectivity. Applications written with the Device Framework can be written to perform exclusively connected (only when a network connection is available), exclusively disconnected (perform all tasks locally on the device) or semi-connected (perform some tasks locally and others using network resources). The tools provided by the abSoapPing library are generally most useful for applications designed to operate in semi-connected mode.

The **SoapPing** object tests for the availability of a network connection by attempting to call a known web service. By default, this object will try to call a specific service published by a known Atoma server. It is also possible to set the object to look for any web service desired in order to verify network availibility.

The Program ID for this control is **absoapping.SoapPing**.

Properties	Methods	Events
SoapPing::Enabled........	SoapPing::Start	SoapPing::OnPingCo mpletion........................
SoapPing::HttpStatus....	SoapPing::Stop..............	
SoapPing::MethodNa me		
SoapPing::Retries		
SoapPing::RouterURL..		
SoapPing::SoapProxy ...		
SoapPing::Succeeded....		
SoapPing::Timeout		

Section 185

SoapPing::Enabled

The **Enabled** property can be used to query to current status of the SoapPing object. While the object is executing ping cycles the value of the Enabled property is true. When the object is idle, the value is false.

```
Visual Basic Syntax
```

boolValue = objSoapPing.Enabled

ABERR_THREADCREATE	An error ocurred while creating the ping

See Also

SoapPing::HttpStatus....
SoapPing::MethodName...............................
SoapPing::ObjectName.................................
SoapPing::OnPingCompletion.......................
SoapPing::Retries.........
SoapPing::RouterURL..
SoapPing::SoapProxy...
SoapPing::Start............
SoapPing::Stop............
SoapPing::Succeeded....
SoapPing::Timeout.......

SECTION 186

SoapPing::HttpStatus

The **HttpStatus** property stores the status code returned by the last request executed by the SoapPing object. As the SoapPing object operates on it's own thread making it difficult to determine precisely when each request has been attempted, this property is generally only accessed at the end of a ping cycle, in the OnPingCompletion event, when determining the status of the network connection.

Visual Basic Syntax

intStatus = objSoapPing.HttpStatus

See Also

SoapPing::Enabled........
SoapPing::MethodNa
me ....
SoapPing::ObjectNam
e ....
SoapPing::OnPingCo
mpletion ....
SoapPing::Retries ....
SoapPing::RouterURL..
SoapPing::SoapProxy ...
SoapPing::Start............
SoapPing::Stop ....
SoapPing::Succeeded....
SoapPing::Timeout.......

Section 187

SoapPing::MethodName

The **MethodName** property specifies the member function that will be called when the destination SOAP router executes the incoming request. For the SoapPing object this method does not perform any processing or data retrieval on the server but simply returns allowing the SoapPing object to determine that the server was successfully reached.

The default value for this property is the function published by the Tx-Sync server object specifically for use with the SoapPing object.

Visual Basic Syntax

objSoapPing.MethodName = strName

See Also

- SoapPing::Enabled........
- SoapPing::HttpStatus....
- SoapPing::ObjectNam
e..................................
- SoapPing::OnPingCo
mpletion......................
- SoapPing::Retries.........
- SoapPing::RouterURL..
- SoapPing::SoapProxy...
- SoapPing::Start............
- SoapPing::Stop............
- SoapPing::Succeeded....
- SoapPing::Timeout.......

SECTION 188

SoapPing::ObjectName

The **ObjectName** property holds the URN of the object that will be called when requests are sent to a server by the SoapPing object. The URN is the identifier given to an object when it is deployed to a SOAP router such that incoming requests can be associated with the object. See the SoapProxy object's ObjectName property for more information.

The default value for this property is the URN of the Atoma system object which provides the core Tx-Sync services.

Visual Basic Syntax

objSoapPing.ObjectName = strName

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me ....
SoapPing::OnPingCo
mpletion ....
SoapPing::Retries ....
SoapPing::RouterURL..
SoapPing::SoapProxy ...
SoapPing::Start.............
SoapPing::Stop ....
SoapPing::Succeeded....
SoapPing::Timeout ....

Section 189

SoapPing::OnPingCompletion

The **OnPingCompletion** event is fired by the SoapPing object to signal that attempts to contact the server have stopped. This event has two possible root causes:

1. a server has been successfully contacted and the network is available
2. no server was contacted before the maximum retries value was reached

The procedure written in your application to handle the OnPingCompletion event can be used to determine the status of the network using the properties of the SoapPing object.

`Visual Basic Syntax`

```
Set objSoapPing = CreateObjectWithEvents("abSoapPing.SoapPing",
"SoapPing_")
...
Sub SoapPing_OnPingCompletion()
...
End Sub
```

See Also

SECTION 190

SoapPing::Retries

The **Retries** property defines the maximum number of attempts the SoapPing Object will make in trying to contact a server. An attempt to contact the server by the SoapPing object consists of a) sending a request to the server and b) pausing for the duration specified in the Timeout property . These two steps form a cycle which is repeated until a) the server is successfully contacted or b) the total repititions of the cycle reaches the number specified by the Retries property. The OnPingCompletion event is triggered when the cycle is stopped at which time the connectivity status can be queried.

The default value of the Retries property is 5.

Visual Basic Syntax

objSoapPing.Retries = intTotal

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me ....
SoapPing::ObjectNam
e ....
SoapPing::OnPingCo
mpletion ....
SoapPing::RouterURL ..
SoapPing::SoapProxy ...
SoapPing::Start............
SoapPing::Stop ....
SoapPing::Succeeded....
SoapPing::Timeout.......

SECTION 191

SoapPing::RouterURL

The **RouterURL** property is used to set the URL of the router that the SoapPing object will use to verify network connectivity. The default value for the RouterURL is the Atoma system router which is used by the device to access Tx-Syncă services. For additional info see the SoapProxy object's RouterURL property .

`Visual Basic Syntax`

objSoapPing.RouterURL = strURL
See Also
SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me ....
SoapPing::ObjectNam
e ....
SoapPing::OnPingCo
mpletion ....
SoapPing::Retries ....
SoapPing::SoapProxy ...
SoapPing::Start.............
SoapPing::Stop ....
SoapPing::Succeeded....
SoapPing::Timeout.......

SECTION 192

SoapPing::SoapProxy

The **SoapProxy** property returns a reference to the instance of the SoapProxy object being used SoapPing object to send and process requests.

`Visual Basic Syntax`

Set objSoapProxy = objSoapPing.SoapProxy

See Also

SECTION 193

SoapPing::Start

The **Start** method activates the SoapPing discovery process. When the Start method is called, the SoapPing object begins the cycle of sending requests for the server at the specified interval in order to verify that network connectivity is availiable.

`Visual Basic Syntax`

SoapPing.Start

ABERR_THREADCREATE	An error ocurred while creating the ping

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me ....
SoapPing::ObjectNam
e ....
SoapPing::OnPingCo
mpletion ....
SoapPing::Retries ....
SoapPing::RouterURL..
SoapPing::SoapProxy ...
SoapPing::Stop ....
SoapPing::Succeeded....
SoapPing::Timeout.......

SECTION 194

SoapPing::Stop

The **Stop** method aborts the network discovery process. When the Stop method is called the SoapPing object stops attempting to contact the remote server. After the Stop method is called, the Start property in order to restart the discovery process.

Visual Basic Syntax

SoapPing.Stop

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me................................
SoapPing::ObjectNam
e..................................
SoapPing::OnPingCo
mpletion.......................
SoapPing::Retries.........
SoapPing::RouterURL..
SoapPing::SoapProxy...
SoapPing::Start............
SoapPing::Succeeded....
SoapPing::Timeout.......

SECTION 195

SoapPing::Succeeded

The **Succeeded** property returns a boolean value indicating the network status discovered by the SoapPing object. If the value is true, a network is available and the server has been contacted. If the value is false, the server was not contacted within the maximum number of attempts specified in the Retries property .

This property should only be checked when responding to the OnPingCompletion event.

```
Visual Basic Syntax
```

boolValue = objSoapPing.Succeeded

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodNa
me................................
SoapPing::ObjectNam
e..................................
SoapPing::OnPingCo
mpletion.......................
SoapPing::Retries.........
SoapPing::RouterURL..
SoapPing::SoapProxy...
SoapPing::Start............
SoapPing::Stop............
SoapPing::Timeout.......

SECTION 196

SoapPing::Timeout

The **Timeout** property defines the interval in seconds between attempts to communicate with a server. After each unsuccessful attempt to reach the desired web service, the SoapPing object will wait for the period specified in the Timeout property before again trying to reach the server. The SoapPing object continues attempting to reach the server in this fashion until the maximum number of attempts specified in the Retries property is reached or the server is successfully contacted.

The default timeout value is 3 seconds.

Visual Basic Syntax

objSoapPing.Timeout = intSeconds

See Also

SoapPing::Enabled........
SoapPing::HttpStatus....
SoapPing::MethodName................................
SoapPing::ObjectName...................................
SoapPing::OnPingCompletion.......................
SoapPing::Retries.........
SoapPing::RouterURL..
SoapPing::SoapProxy...
SoapPing::Start.............
SoapPing::Stop.............
SoapPing::Succeeded....

CHAPTER 13

abStdio (Serial, Irda, Printer)

The **abstdio** library allows applications to access the Input and Output communications ports that are supported by the device operating system. The **abstdio** library defines each communication method as a Transport. The library currently provides Serial and IrDa objects that enable the use of the COMM and InfraRed ports on suitably equipped devices.

Additionally, the library provides a Printer object that makes communication with mobile printers virtually transparent to an application. An application can use any communications mechanism supported by the model of printer assigned to transmit the information to be printed. The Printer object provides a set of methods and properties that simplify the printing of barcode labels and receipts.

The **Irda** control allows an application to send and receive data via the infrared port of a client device. The **Irda** control provides properties to configure the communication settings of the infrared port of the device and methods to send and receive data.

The Program ID for this object is **abstdio.IrDa**.

Properties	Methods	Events
Irda::Config..................	Irda::CancelRead...........	Irda::OnConnect ....
Irda::DataAvailable.......	Irda::Connect	Irda::OnDataAvailable ..
Irda::DeviceName.........	Irda::Disconnect............	Irda::OnDisconnect.......
Irda::DiscoveryTimeout..................................	Irda::GetData	
Irda::Enabled	Irda::Read	
Irda::InputBufferSize....	Irda::Write	
Irda::IsConnected..........		
Irda::IsReading		
Irda::ReadMode............		
MSR::Separator............		
Irda::ServiceName........		

SECTION 197

Irda::CancelRead

The **CancelRead** method stops aysnchronous reading on the IrDa port. When the **IrDa** object is set to read asynchronously, the OnDataAvailable (Irda) event is fired when information is received by the port. A call to the CancelRead method causes the object to abort reading from the port and correspondingly to stop firing the OnDataAvailable event.

Visual Basic Syntax

objIrDa.CancelRead

See Also

Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

Section 198

Irda::Config

The **Config** property is a read/write property that returns or sets the configuration scheme for the IrDa object. A configuration scheme consists of the values assigned to the DiscoveryTimeout , InputBufferSize , PrintMode and ServiceName properties. To set the values supplied in a configuration sheme, an application should use the corresponding properties of the IrDa object.

Visual Basic Syntax

Set objIrDa.Config = objConfig

Error Values

ABERR_GETCONFIG	An error ocurred while creating the configurat
ABERR_INVALIDCONFIG	Invalid configuration object.
ABERR_PUTCONFIG	An error ocurred while setting the configuratic

See Also

Irda::CancelRead..........
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 199

Irda::Connect

The **Connect** method opens a communication session using the IrDa port. In order to open a communication session, the IrDa object attempts to discover any devices within the range of the infrared port and to open a connection with the first device that responds to the discovery. The IrDa object will raise an error if it is uable to connect to a device when the Connect method is called.

If desired the ServiceName and PrintMode properties can be used to make specific reference to the type connection desired.

Visual Basic Syntax

objIrDa.Connect

Error Values

ABERR_IRDA_SOCKCREATE	An error ocurred while creating the Irda s
ABERR_IRDA_DISCOVERTIMEOUT	Failed to connect to an Irda device.

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect ....
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled ....
Irda::GetData ....
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading ....
Irda::OnConnect ....
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode ....
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 200

Irda::DataAvailable

The **DataAvailable** property returns a boolean value indicating whether information has been received on the IrDa port or not. If the value is true, data has been received by the infrared port. If the value is false, no data has been received. If data has been received by the IrDa object, the data can be accessed by calling the GetData (Irda) method.

`Visual Basic Syntax`

boolValue = objIrDa.DataAvailable
See Also
Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut..................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected.........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 201

Irda::DeviceName

The **DeviceName** property returns the "friendly" name of the device with which the IrDa object most recently connected. The DeviceName is a text text string received from a device during the discovery process. If the IrDa object is currently connected, the DeviceName property will represent the name of the currently connected device.

Visual Basic Syntax

```
strName = objIrDa.DeviceName
```

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected.........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

Section 202

Irda::Disconnect

The **Disconnect** method closes an open infrared connection. A call to the Disconnect method will cause and reading or writing to the port to be aborted and the port to be closed by the application. An infrared connection can also be closed by setting the Enabled (IrDA) property to false.

```
Visual Basic Syntax
```

objIrda.Disconnect

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected.........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 203

Irda::DiscoveryTimeout

The **DiscoveryTimeout** period is a value measured in seconds that determines the total length of time that the IrDa object will spend polling for devices when the Connect (IrDa) method is called. The default value for this property is 10. After the Connect method is called, if the IrDa object is unable to locate a device within the period specified by the DiscoveryTimeout property, the Connect method will fail.

The DiscoveryTimeout property should be set before the Connect method of the object is called or the object is enabled.

Visual Basic Syntax

objIrDa.DiscoveryTimeout

Error Values

ABERR_INVALIDTIMEOUT	Invalid timeout value.

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 204

Irda::Enabled

The **Enabled** property can be used to open and close the infrared port much in the same way that the Connect (IrDa) and Disconnect (IrDa) methods are used. It can also be used to query the status of the IrDa object.

To open an infrared connection, set the value of the Enabled property to true. If the IrDa object is unable to open a connection it will raise an error. To close an infrared connection, set the value of the Enabled property to false.

Visual Basic Syntax

objIrDa.Enabled = boolValue

Error Values

ABERR_IRDA_SOCKCREATE	An error ocurred while creating the Irda s
ABERR_IRDA_DISCOVERTIMEOUT	Failed to connect to an Irda device.

See Also

SECTION 205

Irda::GetData

The **GetData** method returns the data held in the IrDa object's input buffer. Data is inserted in to the IrDa object's input buffer as information is received on the infrared port. This method returns a string value equivalent to the data currently held in the buffer. After this method is called, the data is removed from the IrDa object's input buffer.

When the Read (Irda) method is called, data is appended to the IrDa object's buffer as it is received. The GetData method can be called after a call to the Read method to retrieve the data.

Visual Basic Syntax

strValue = objIrda.GetData

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeout...............................
Irda::Enabled................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 206

Irda::InputBufferSize

The **InputBufferSize** property is a read/write property that sets or retrieves the size of the infrared port's input buffer. This property determines the maximum number of characters that can be read from the infrared port during a single read operation. This property must be set before the IrDa object is enabled.

During synchronous reading, the IrDa object performs a single read operation from the infrared port when the Read (Irda) method is called. The maxmimum number of characters that can be received with this single read operation is equal to the InputBufferSize. When reading asynchronously, the IrDa object may perform multiple read operations. As each read operation is completed, data is moved from the infrared port's input buffer and appended to a second buffer controlled by the IrDa object. This second buffer is not limited by the InputBufferSize setting.

The default value for this property is 512.

Visual Basic Syntax

objIrDa.InputBufferSize = intValue

Error Values

ABERR_INVALIDBUFFERSIZE	Invalid buffer size.

See Also

Irda::CancelRead ....
Irda::Config ....
Irda::Connect ....
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect ....
Irda::DiscoveryTimeo
ut ....
Irda::Enabled ....
Irda::GetData ....
Irda::IsConnected..........
Irda::IsReading ....
Irda::OnConnect ....
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode ....
Irda::Read ....
Irda::ReadMode............
Irda::ServiceName........
Irda::Write ....

SECTION 207

Irda::IsConnected

The **IsConnected** property returns a value that inidcates the status of the infrared port. If the value is **true**, the infrared port currently has an open connection. If the value is **false**, the infrared port does not have an open connection with another device.

Visual Basic Syntax

boolValue = objIrDa.IsConnected

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut..................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

SECTION 208

Irda::IsReading

The **IsReading** property returns a value indicating the read status of the IrDa object. If the value is **true** the IrDa object is currently reading data from the port. If the value is false, the IrDa object is not reading data from the infrared port. This property is only applicable when reading asynchronously.

Visual Basic Syntax

boolValue = objIrDa.IsReading

See Also

SECTION 209

Irda::OnConnect

The **OnConnect** event is fired when the IrDa object successfully connects to an infrared device. This event is not fired if the IrDa object is unable to connect to a device when the Connect (IrDa) method is called.

```
Visual Basic Syntax
```

Set objIrDa = CreateObjectWithEvents("abstdio.IrDa", "IrDa_")
...
Sub IrDa_OnConnect()
...
End Sub
See Also

Section 210

Irda::OnDataAvailable

The **OnDataAvailable** event is fired when the IrDa object successfully reads data from the infrared port. This event is only fired when reading is performed asynchronously. The GetData method can be used to retrieve the data read from the port as a string.

Visual Basic Syntax

```
Set objIrDa = CreateObjectWithEvents("abstdio.IrDa", "IrDa_")
...
Sub IrDa_OnDataAvailable()
...
End Sub
```

See Also

SECTION 211

Irda::OnDisconnect

The **OnDisconnect** event is fired when and infrared connection is closed by the IrDa object. If there is an open connection on the infrared port that is interrupted for any reason, the OnDisconnect event will be fired to notify an application that the connection has been closed.

Visual Basic Syntax

```
Set objIrDa = CreateObjectWithEvents("abstdio.IrDa", "IrDa_")
...
Sub IrDa_OnDisconnect()
...
End Sub
```

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected.........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

```
objPrinter.StartDoc
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead.
Printer::HumanReadable................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X.....................
Printer::Y.....................

SECTION 212

Irda::PrintMode

The **PrintMode** property allows an application to set the type of infrared connnection that will be accepted by the IrDa object. The infrared port is capable of using a communications layer known as *IrLPT* that is supported by many printers equipped with infrared ports. When the PrintMode property is set to **true**, the IrDa object will use the IrLPT protocol during infrared communications. While in this mode, the IrDa object will only communicate with devices that support the IrLPT protocol. When the PrintMode property is set to **false**, the IrDa object will use the *IrLMP* protocol and the types of connections supported by the IrDa object can be further filtered using the ServiceName property.

If an application wishes to change the PrintMode, this property should be set before the Connect (IrDa) method is called.

Refer to the device operating system documentation or the Infrared Data Association "http://www.irda.org" for more information on the IrDA specifications.

```
Visual Basic Syntax
```
objIrDa.PrintMode = boolValue

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::Read.....................
Irda::ReadMode............
Irda::ServiceName........
Irda::Write....................

Section 213

Irda::Read

The **Read** method is used to start recieving infromation from the infrared port. This method takes two parameters. The first parameter, ReadMode, is a value indicating the type of read operation to be executed. The second parameter, Seconds, specifies the length of time that a synchronous read operation will be executed.

The possible values of the ReadMode parameter are defined in the stdioREADMODE enumeration. The two possible types of read operations are synchronous and asynchronous. When a synchronous read is selected, the IrDa object performs a single read operation. When this read operation is completed, the Read method returns with the data read from the port assigned to the output value of the method.

When an asynchronous read is selected, the IrDa object will begin performing read operations indefinitely and the Read method will return immediately. The OnDataAvailable (Irda) event will be fired as each successful read operation is completed. The CancelRead method can be used to stop asynchronous reading from the infrared port. When an asynchronous read is specified, the read method will not return any values read from the infrared port.

```
Visual Basic Syntax
```

```
strValue =objIrda.Read( stdioReadMode, intSeconds )
```

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::ReadMode............
Irda::ServiceName......,.
Irda::Write....................

SECTION 214

Irda::ReadMode

The **ReadMode** property returns the current read setting of the IrDa object. The possible values returned by the ReadMode property are specified in the stdioREADMODE enumeration. The possible read settings of the IrDa object are *synchronous* and *asynchronous*. See the Read (Irda) method for more information on the types of read settings.

```
Visual Basic Syntax
```

intValue = objIrDa.ReadMode

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected.........
Irda::IsReading............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ServiceName........
Irda::Write....................

SECTION 215

Irda::ServiceName

The **ServiceName** property is used to limit the type of connection that the IrDa object will establish. When establishing a connection with an IrDA device, the IrDa specification allows each device or station to expose multiple endpoints or services. The Service Name property can be used to directly specify the name of service that the IrDa object will attempt to connect to when a device is located. The default value for this property is *LSAP-SEL1*.

When the PrintMode property is enabled the ServiceName property will not be used when connecting to a device. When PrintMode is enabled the IrDa object will only attempt to connect to devices supporting *IrLPT*.

Refer to the device operating system documentation or the Infrared Data Association "http://www.irda.org" for more information on the IrDA specifications.

```
Visual Basic Syntax
```

objIrDa.ServiceName = strValue

See Also

Irda::CancelRead..........
Irda::Config..................
Irda::Connect................
Irda::DataAvailable.......
Irda::DeviceName.........
Irda::Disconnect...........
Irda::DiscoveryTimeo
ut.................................
Irda::Enabled................
Irda::GetData................
Irda::InputBufferSize....
Irda::IsConnected..........
Irda::IsReading.............
Irda::OnConnect...........
Irda::OnDataAvailable..
Irda::OnDisconnect.......
Irda::PrintMode............
Irda::Read.....................
Irda::ReadMode............
Irda::Write....................

SECTION 216

Irda::Write

The **Write** method is used to send data using the infrared port. The Write method has one string parameter that holds the data to be sent using the IrDa object. An infrared connection must be opened using the Connect (IrDa) method or Enabled (IrDA) property before data can be successfully written using the port.

If the IrDa object is not able to write the data an error is thrown.

`Visual Basic Syntax`

objIrDa.Write(strData)

`Error Values`

ABERR_IRDA_SEND	An error ocurred while writing to the Irda

See Also

The **Printer** object can be used to print information directly from your application to a variety of printers. The methods and properties of the object allow straightforward printing of text and barcode data on printers using any of the transports defined by the **abstdio** library.

The Printer object allows you to manipulate the lines of information and associate them with a printer document. The document is the set of printer commands constructed internally by the Printer object that will be transmitted to the printer device on command. The print position determines where a line of information will appear in the printer output. This position can be set directly at any time and is also adjusted automatically as lines of information are inserted into a document.

The Printer object provides the functionality most commonly found accross different brands of printers along with a few featrues that are supported on some printers and not on others. If your application targets multiple models of printers from different manufacturers it is always best to test the application using the desired printer models as there may be slight variations in the resulting printer output due to differences in the capabilities and feature selection of the supported printers. This documentation tries to point out these differences where possible however you should consult the documentation provided by your printer manufacturer for exact specifications and information on performing more complex printing operations.

The Program ID for this object is **abstdio.Printer.**

`Printer Support`

Zebra Cameo2

Zebra Encore2

Oneill Microflash 2tcr

Properties	Methods	Events
Printer::Barcode	Printer::Advance	Printer::OnDataAvailable
Printer::BarcodeName	Printer::CancelHardwareRead	
Printer::Copies	Printer::DrawLine	
Printer::DataAvailable	Printer::EndDoc	
Printer::Font	Printer::GetHardwareData	
Printer::FontName	Printer::HardwareRead	
Printer::HumanReadable	Printer::PrintBarcode	
Printer::Indentation	Printer::PrintMaxicode	
Printer::IsReading	Printer::PrintPDF417	
Printer::LabelHeight	Printer::PrintText	
Printer::LabelWidth	Printer::StartDoc	
Printer::Orientation		
Printer::PrinterType		
Printer::PrintSize		
Printer::ReadMode		
Printer::Spacing		
Printer::Transport		
Printer::TransportConfig		
Printer::X		
Printer::Y		

Section 217

Printer::Advance

The **Advance** method moves the current print position down the vertical axis of the printer output. This method takes one parameter indicating the number of output lines to skip when moving the print position. The height of each line skipped varies depending on the settings of the Font and PrintSize when the Advance method is called. The height of each line skipped is roughly equivalent to the vertical height of the currently selected font.

The Advance method can only be executed between calls to the StartDoc and EndDoc methods and in this way can be easily used to insert blank lines into the printer output.

```
Visual Basic Syntax
```

```
objPrinter.Advance intValue
```

See Also

Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing...........
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X.....................
Printer::Y.....................

SECTION 218

Printer::Barcode

The **Barcode** property sets the current barcode symbology. To set a symbology assign the integer value corresponding to the symbology to the Barcode property. The possible values and their corresponding symbologies are defined in the stdioBARCODE enumeration. The text passed to the PrintBarcode method is encoded with the symbology specified by the Barcode property. The default symbology is Codė 39.

```
Visual Basic Syntax
```

```
objPrinter.Barcode = iSymbology
```

See Also

SECTION 219

Printer::BarcodeName

The **BarcodeName** property returns the text name of the barcode symbology that is currently designated by the Barcode property. The BarcodeName property is a read-only property. The symbology can only be set by assigning the appropriate value to the Barcode property.

Visual Basic Syntax

```
strName = objPrinter.BarcodeName
```

Error Values

ABERR_PRINTER_NOTLOADED	Printer type has not been selected and loa
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_PRINTER_GETBARCODENAME	Unable to retrieve barcode name.
ABERR_PRINTER_LOADXML	Unable to load XML document.

See Also

Printer::Advance...........
Printer::Barcode............
Printer::CancelHardwa
reRead..........................
Printer::Copies..............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardware
Data..............................
Printer::HardwareRead.
Printer::HumanReadab
le..................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble................................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport.........
Printer::TransportConf
ig..................................

Printer::X......................
Printer::Y......................

SECTION 220

Printer::CancelHardwareRead

The **CancelHardwareRead** method aborts reading from the printer on the current transport . This method should be used only with asynchronous reading . Refer to the printer vendor documentation for more information about the capabilities of the mobile printer being used.

Visual BasicSyntax

```
objPrinter.CancelHardwareRead
```

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_NOTLOADED	Printer type has not been selected and loa
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::Copies.............
Printer::DataAvailable ..
Printer::DrawLine........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName........
Printer::GetHardware
Data............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth......
Printer::OnDataAvaila
ble ....
Printer::Orientation.......
Printer::PrintBarcode ....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 221

Printer::Copies

The **Copies** property sets the number of times the current document will be printed when the document is completed. When printing to mobile receipt printers it is usually necessary to insert a few blank lines at the end of the document when printing multiple copies to create a separation between the copies.

`Visual Basic Syntax`

objPrinter.Copies = intValue

`Error Values`

ABERR_INVALIDCOPIES	Invalid number of copies.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead.........................
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font.................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead.
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X.....................
Printer::Y.....................

SECTION 222

Printer::DataAvailable

The **DataAvailable** property returns a boolean value indicating whether data has been read by a printer accessory. If the value is True, data has been read from the printer. If the value is false no data is available. To retrieve the information that was read from the printer, use the GetHardwareData method of the Printer object.

```
Visual Basic Syntax
```

```
boolValue = objPrinter.DataAvailable
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead.........................
Printer::Copies.............
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardwareData.............................
Printer::HardwareRead .
Printer::HumanReadable ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth......
Printer::OnDataAvailable ....
Printer::Orientation.......
Printer::PrintBarcode ....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing ....
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X......................
Printer::Y......................

SECTION 223

Printer::DrawLine

The **DrawLine** method can be used to add borders to the printer output. The DrawLine method accepts four parameters. The first two parameters *x0* and *y0* indicate the starting coordinates of the line. The starting coordinates are optional and will default to the current print position. The last two parameters *x1* and *y1* indicate the ending coordinates of the line. The width of the line is determined by the current value of the PrintSize property.

To draw a horizontal line, the x-coordinate that specifies the end position (*x1*) is used to indicate the width of the line. The width of the line will be equal to the difference between the x-coordinate of the current print position and the x-coordinate of the ending position specified. For a horizontal line, the y-coordinate that specifies the end position (*y1*) should remain unchanged. This is accomplished by assigning the value -1 to the *y1* parameter.

To draw a vertical line, the y-coordinate that specifies the end position (*y1*) is used to indicate the height of the line. The height of the line will be equal to the difference between the y-coordinate of the current print position and the y-coordinate of the ending position specified. For a vertical line, the x-coordinate that specifies the end position (*x1*) should remain unchanged. This is accomplished by assigning the value -1 to the *x1* parameter. After a vertical line is drawn, the print position remains approximately where the line was started therefore it may be necessary to explicitly specify the position of any further text printed to avoid overlapping items in the print output.

`Visual Basic Syntax`

objPrinter.DrawLine [intX0], [intY0], intX1, intY1

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw.........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X.....................
Printer::Y.....................

SECTION 224

Printer::EndDoc

The **EndDoc** method closes a printer session that was opened with the StartDoc method and sends the information to be printed over the selected Transport . The EndDoc method must always be preceeded by a call to the StartDoc method. An error is raised if the Printer object is unable to send data to the printer when the EndDoc method is called.

Visual Basic Syntax

objPrinter.EndDoc

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_NOTLOADED	Printer type has not been selected and loa
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_PRINTER_LOADXML	Unable to load XML document.
ABERR_STARTDOC	StartDoc has not been called.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::Font................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead .
Printer::HumanReadable................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........

SECTION 225

Printer::Font

The **Font** property is a read/write property that determines the the style and appearance of the characters printed when the PrintText method is called. The stdioTEXTFONT enumeration lists the possible values for the Font property. To specify a font, set the Font property to the corresponding value from the stdioTEXTFONT enumeration. Each subsequent calls to the PrintText method will use the specified font.

The printers supported by the Printer object support a limited selection of fonts. The actual printed font corresponding to each value of the stdioTEXTFONT enumeration varies between the different brands of printers supported.

```
Visual Basic Syntax
```

```
objPrinter.Font = intValue
```

See Also

Printer::Advance...........
Printer::Barcode............
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies..............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead .
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 226

Printer::FontName

The **FontName** property returns name of the currently selected printer font. Printer fonts can be selected through the Font property of the Printer object. The name returned by the FontName property depends on the brand of printer being used and the selection of fonts available on the printer. Consult your printer documentation for more information on the available fonts.

Visual Basic Syntax

strValue = objPrinter.FontName

Error Values

ABERR_PRINTER_NOTLOADED	Printer type has not been selected and loa
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_PRINTER_GETFONTNAME	Unable to retrieve font name.
ABERR_PRINTER_GETBARCODENAME	Unable to retrieve barcode name.
ABERR_PRINTER_LOADXML	Unable to load XML document.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble..............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw.........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc..........
Printer::Transport..........

SECTION 227

Printer::GetHardwareData

The **GetHardwareData** method returns the data held in the Printer object's input buffer. Data is inserted into the Printer object's input buffer as information is received from the printer hardware. This method returns a string value equivalent to the data currently held in the buffer. After this method is called, the data is removed from the Printer object's input buffer. This method should only be used with printers that are capable of receiving input such as printers equipped with magnetic stripe readers.

When the HardwareRead method is called, data is appended to the Printer object's buffer as it is received. The GetHardwareData method can be called after a call to the HardwareRead method to retrieve the data.

```
Visual Basic Syntax
```

```
strValue = objPrinter.GetHardwareData
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::HardwareRead .
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing...........
Printer::StartDoc..........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X.....................
Printer::Y.....................

SECTION 228

Printer::HardwareRead

The **HardwareRead** method is used to start recieving infromation from a printer. This method takes three parameters. The first parameter, HardwareType, is a value indicating the type of auxilliary printer hardware that will be used to perform the read operation. The second parameter, ReadMode, is a value indicating the type of read operation to be executed. The third parameter, Seconds, specifies the length of time that a synchronous read operation will be executed.

The possible values of the HardwareType parameter are defined in the stdioHARDWARETYPE enumeration. The value selected determines the method used by the printer object to retrieve data from the printer hardware. For many read operations, the HardwareRead method sends commands to the printer hardware to activate the hardware for reading. The application user must then physically trigger the read (e.g. swipe a magnetic stripe card or point a scanner laser at a barcode) before the printer object receives any data.

The possible values of the ReadMode parameter are defined in the stdioREADMODE enumeration. The two possible types of read operations are synchronous and asynchronous. When a synchronous read is selected, the Printer object performs a single read operation. When this read operation is completed, the Read method returns with the data read from the port assigned to the output value of the method.

When an asynchronous read is selected, the Printer object will begin performing read operations indefinitely and the Read method will return immediately. The OnDataAvailable (Printer) event will be fired as each successful read operation is completed. The CancelHardwareRead method can be used to stop asynchronous reading from the printer. When an asynchronous read is specified, the Read method will not return any values read from the printer.

`Visual Basic Syntax`

```
strValue = objPrinter.HardwareRead( HardwareType, ReadMode ,
[intSeconds] )
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font.................
Printer::FontName........
Printer::GetHardwareData..............................
Printer::HumanReadable..................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X......................
Printer::Y.....................

Section 229

Printer::HumanReadable

The **HumanReadable** property is a boolean value used to indicate whether barcode data should be accompanied by a text representation of the encoded data. When the HumanReadable property is set to **true**, any data printed with the PrintBarcode method will also be printed in human readable text below the barcode. The size of the text is determined by the current setting of the PrintSize property. When the HumanReadable property is set to **false**, the text representation of encoded barcode data is not automatically printed.

```
Visual Basic Syntax
```

objPrinter.HumanReadable = boolValue

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font.................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 230

Printer::Indentation

The **Identation** property is a read/write property that specifies the current left margin in the printer output. When printing with a horizontal orientation, this value indicates the default starting x-coordinate position of each line printed using the PrintText , PrintBarcode , PrintMaxicode or PrintPDF417 methods. The Indentation property can be set at any point between matching calls to the StartDoc and EndDoc methods. This value is measured in printer dots. Consult your printer documentation for information regarding the dot-size of your printer.

```
Visual Basic Syntax
```

```
objPrinter.Indentation = intValue
```

Error Values

ABERR_INVALIDINDENTATI ON	Invalid indentation value.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardware Data............................
Printer::HardwareRead .
Printer::HumanReadab le.................................
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw.........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf ig................................
Printer::X......................
Printer::Y......................

SECTION 231

Printer::IsReading

The **IsReading** property returns a value indicating the read status of the Printer object. If the value is **true** the Printer object is currently reading data from a printer's auxilliary hardware. If the value is false, the Printer object is not reading data from the printer. This property is only applicable when reading asynchronously.

```
Visual Basic Syntax
```

```
boolValue = objPrinter.IsReading
```

See Also

Printer::Advance...........
Printer::Barcode............
Printer::BarcodeName...
Printer::CancelHardwareRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead.
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X......................
Printer::Y......................

Section 232

Printer::LabelHeight

The **LabelHeight** property sets the length, in printer dots, of the print output. For printers that use a continuously fed media, such as receipt printers, the LabelHeight property sets the physical height of the output printed when the EndDoc method is called. The printer stops printing when the vertical length of the information printed reaches the value of the LabelHeight property. The amount of barcode or text information to be printed does not affect the length of the printer output. The printer will print as much barcode or text data as it can before the height of the output equals the specified label height. If no barcode, text or line data is specified before the EndDoc method is called, the printer will print a "blank label" i.e. the print position will advance by a value equal to the LabelHeight.

`Visual Basic Syntax`

objPrinter.LabelHeight = intValue

`Error Values`

ABERR_INVALIDLABELHEIG HT	Invalid label height.

See Also

Printer::Advance...........
Printer::Barcode............
Printer::BarcodeName...
Printer::CancelHardwa reRead..........................
Printer::Copies..............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardware Data.............................
Printer::HardwareRead.
Printer::HumanReadab le..................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelWidth......
Printer::OnDataAvaila ble................................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf ig.................................
Printer::X......................
Printer::Y......................

SECTION 233

Printer::LabelWidth

The **LabelWidth** property sets the horizontal size, in printer dots, of the printer output. On many printers, this value does not affect the actual size of the information printed. Typically the printer will print a line of information until the edge of the printer media is encountered, regardless of the value of the LabelWidth property. It is generally not necessary to set the value of this property.

`Visual Basic Syntax`

objPrinter.LabelWidth = intValue

`Error Values`

ABERR_INVALIDLABELWIDTH	Invalid label width.

See Also

Printer::Advance...........
Printer::Barcode............
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies..............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 234

Printer::OnDataAvailable

The **OnDataAvailable** event is fired when the printer hardware successfully reads data during a HardwareRead operation. This event is only fired when reading is performed asynchronously. The GetHardwareData method can be used to retrieve the data read from the printer as a string.

```
Visual Basic Syntax
```

```
Set objPrinter = CreateObjectWithEvents("abstdio.Printer", "Printer_")
...
Sub Printer_OnDataAvailable()
...
End Sub
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead.
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X......................
Printer::Y......................

SECTION 235

Printer::Orientation

The **Orientation** property is a read/write property that sets the direction in which data is printed. The possible values of the Orientation property are listed in the stdioORIENTATION enumeration. There are two options for the print orientation: **horizontal** and **vertical**. The default value for this property is **horizontal**. When printing vertically, the physical interpretation of the X and Y coordinates varies by printer.

```
Visual Basic Syntax
```

objPrinter.Orientation = intValue

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead ....
Printer::Copies.............
Printer::DataAvailable ..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardware
Data.............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth ......
Printer::OnDataAvaila
ble ....
Printer::PrintBarcode ....
Printer::PrinterType ......
Printer::PrintMaxicode..
Printer::PrintPDF417 ....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 236

Printer::PrintBarcode

The **PrintBarcode** method adds a line of data, encoded as a barcode, to the print output. This method has three possible parameters. The first parameter is the data that will be encoded in the barcode. This data will be encoded using the current symbology specified by the Barcode property. Some barcode symbologies are designed to encode a specific subset of all possible characters. For example, the *UPC* symbology only encodes numeric characters (0 through 9). If the current symbology cannot encode the text data specified in the first parameter, the barcode will not appear in the print output and/or the printer may fail to print any of the data added to the output.

The second parameter is the x-coordinate where the barcode will be printed. The third parameter is the y-coordinate where the barcode will be printed. These coordinates are optional and if they are not specified, the barcode will be printed at the current print position. The coordinates are specified in printer dots.

This method should only be used between calls to the StartDoc and EndDoc methods.

Visual Basic Syntax

objPrinter.PrintBarcode strText, [intX], [intY]

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_STARTDOC	StartDoc has not been called.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardware
Data............................
Printer::HardwareRead.
Printer::HumanReadab
le................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble..............................
Printer::Orientation.......
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport.........
Printer::TransportConf
ig................................
Printer::X.....................
Printer::Y.....................

SECTION 237

Printer::PrinterType

The **PrinterType** property is a read/write property that identifies the manufacturer and model of printer being used. The string value assigned to the PrinterType property is used to determine how the printer object will communicate with the printer hardware. The PrinterType property should be set before any other methods or properties of the Printer object are called. The values supported are "Cameo2", "Encore2" and "Microflash2tcr". This property is case-sensitive.

Visual Basic Syntax

objPrinter.PrinterType= strValue

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_NOTLOADED	Printer type has not been selected and loaded.
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printer settings.
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new label document.
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_PRINTER_LOADXML	Unable to load XML document.

See Also

Printer::Advance...........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwareRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardwareData.............................
Printer::HardwareRead .
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............

SECTION 238

Printer::PrintMaxicode

The **PrintMaxicode** method adds data, encoded using the Maxicode symbology, to the print output. This method has six possible parameters. The first four parameters are required in order to successfully print the Maxicode barcode. These parameters are the Postal Code, Country Code, Service Class and Low Priority Message associated with the Maxicode specification. Calls to the PrintMaxicode method have no effect on the current symbology specified by the Barcode property. The Barcode property only determines the symbology when the PrintBarcode method is used.

The final two parameters specify the x-coordinate and y-coordinate where the barcode will be printed. These coordinates are optional and if they are not specified, the barcode will be printed at the current print position. A value of -1 can be passed through these parameters to indicate that the current X and Y position should be used. The coordinates are specified in printer dots.

This method should only be used between calls to the StartDoc and EndDoc methods.

Visual Basic Syntax

```
objPrinter.PrintMaxicode( strPostalCode, strCountryCode,
strClass, strMessage, [intX], [intY] )
```

Error Values

SECTION 239

Printer::PrintPDF417

The **PrintPDF417** method adds data, encoded using the PDF-417 symbology, to the print output. This method has seven possible parameters. The first five parameters are required in order to successfully print a PDF-417 barcode. These parameters are the Unit Width, Unit Height, Columns, Security Level and Text Message associated with the PDF-417 specification. Calls to the PrintPDF417 method have no effect on the current symbology specified by the Barcode property. The Barcode property only determines the symbology when the PrintBarcode method is used.

The final two parameters specify the x-coordinate and y-coordinate where the barcode will be printed. These coordinates are optional and if they are not specified, the barcode will be printed at the current print position. A value of -1 can be passed through these parameters to indicate that the current X and Y position should be used. The coordinates are specified in printer dots.

This method should only be used between calls to the StartDoc and EndDoc methods.

Visual Basic Syntax

```
objPrinter.PrintPDF417 intWidth, intHeight, intColumns,
intSecurity, strText, [intX], [intY]
```

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_STARTDOC	StartDoc has not been called.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable ..
Printer::DrawLine........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardware
Data............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading........
Printer::LabelHeight .....
Printer::LabelWidth......
Printer::OnDataAvaila
ble ....
Printer::Orientation.......
Printer::PrintBarcode ....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport.........
Printer::TransportConf
ig.................................
Printer::X.....................
Printer::Y.....................

SECTION 240

Printer::PrintRaw

The **PrintRaw** method is used to insert printer language commands into the data stream sent to the printer. The PrintRaw method has one parameter that is a string containing the printer language commands to be inserted. The command(s) included in this string are not interpreted by the printer object and are passed directly to the printer without modification. The PrintRaw method is useful for achieving printer functionality not provided with the other methods and properties of the Printer object.

The text passed to the PrintRaw method will be added to the sequence of commands automatically generated when the StartDoc , EndDoc and any other Printer object methods are called. Because of this it is important to consider how the command(s) specified with the text parameter will interact with the automatically generated commands. If you wish to take complete control of all aspects of communication with a printer consider using the Serial or IrDa objects.

Visual Basic Syntax

```
objPrinter.PrintRaw strText
```

Error Values

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_STARTDOC	StartDoc has not been called.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine........
Printer::EndDoc...........
Printer::Font................
Printer::FontName........
Printer::GetHardware
Data............................
Printer::HardwareRead.
Printer::HumanReadab
le................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing...........
Printer::StartDoc...........
Printer::Transport.........
Printer::TransportConf

Section 241

Printer::PrintSize

The **PrintSize** property is a read/write property that specifies the size of the printing fonts and the thickness of the drawing pen. The stdioPRINTSIZE enumeration lists the possible values of the PrintSize property. This property determines the size of data printed using the PrintText , PrintBarcode , PrintMaxicode , PrintPDF417 and DrawLine methods.

`Visual Basic Syntax`

```
objPrinter.PrintSize = intValue
```

See Also

Printer::Advance...........
Printer::Barcode............
Printer::BarcodeName...
Printer::CancelHardwareRead..........................
Printer::Copies..............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardwareData..............................
Printer::HardwareRead.
Printer::HumanReadable.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvailable................................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConfig.................................
Printer::X......................
Printer::Y......................

SECTION 242

Printer::PrintText

The **PrintText** method adds a line of text information to the print output. This method has three possible parameters. The first parameter is the text information that will appear in the print output. This text will be printed using the current font specified by the Font property. Most mobile printers only support the ASCII character set so the text printed is usually limited to ASCII printable characters.

The second parameter is the x-coordinate where the text line will be printed. The third parameter is the y-coordinate where the text line will be printed. These coordinates are optional and if they are not specified, the text line will be printed at the current print position. The X and Y coordinates are specified in printer dots. A carriage return is automatically added to the line of text printed and the current print position is advanced to the next line.

This method should only be used between calls to the StartDoc and EndDoc methods.

`Visual Basic Syntax`

objPrinter.PrintText strText, [intX], [intY]

`Error Values`

ABERR_PRINTER_NOTRANSPORT	Printer transport has not been loaded.
ABERR_PRINTER_SETTINGS	An error ocurred while loading the printe
ABERR_PRINTER_NEWLABEL	An error ocurred while loading a new lab
ABERR_PRINTER_LABELNOTLOADED	A label document has not been loaded.
ABERR_STARTDOC	StartDoc has not been called.

See Also

SECTION 243

Printer::ReadMode

The **ReadMode** property returns the current hardware read setting of the Printer object. The possible values returned by the ReadMode property are specified in the stdioREADMODE enumeration. The possible hardware read settings of the Printer object are *synchronous* and *asynchronous*. See the HardwareRead method for more information on the types of read settings.

```
Visual Basic Syntax
```

```
intValue = objPrinter.ReadMode
```

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies.............
Printer::DataAvailable ..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardware
Data.............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth ......
Printer::OnDataAvaila
ble ....
Printer::Orientation.......
Printer::PrintBarcode ....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 244

Printer::Spacing

The **Spacing** property is a read/write property that specifies the size of the "whitespace" inserted between lines in the printer output. When the PrintText , PrintBarcode , PrintMaxicode or PrintPDF417 methods are used, a line in the printer output is created and the printer position is advanced such that the next call to one of these methods will create a new line instead of overwriting the previously inserted line. The spacing property defines the separation between these lines. This property is measured in printer dots and the default value is 0.

Visual Basic Syntax

objPrinter.Spacing = intValue

Error Values

ABERR_INVALIDSPACING	Invalid spacing value.

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead.........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardware
Data............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth ......
Printer::OnDataAvaila
ble ....
Printer::Orientation.......
Printer::PrintBarcode ....
Printer::PrinterType ......
Printer::PrintMaxicode..
Printer::PrintPDF417 ....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 245

Printer::StartDoc

The **StartDoc** method is used to specify the start of a sequence of printer commands that will be sent to a printer as a group. The end of the sequence of commands is specified using the EndDoc method. When the StartDoc method is called, the properties of the Printer object are initialized to their default values and any pending or previous printer output commands are cleared. The PrinterType property must be set before this method is called.

```
Visual Basic Syntax
```

SECTION 246

Printer::Transport

The **Transport** property is a read/write property that specifies the communications mechanism used to transmit and receive data from a printer. The stdioTRANSPORT enumeration lists the possible values of the Transport property. When the EndDoc or HardwareRead method is called, the Printer object attempts to communicate with a printer using the communications mechanism specified in the Transport property. When the Printer object is first created, the default Transport property value specifies serial communications (*stdioTRANSPORT_SERIAL*).

```
Visual Basic Syntax
```

objPrinter.Transport = intValue

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead ....
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc...........
Printer::Font ....
Printer::FontName ....
Printer::GetHardware
Data.............................
Printer::HardwareRead .
Printer::HumanReadab
le ....
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight .....
Printer::LabelWidth......
Printer::OnDataAvaila
ble ....
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize ....
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing ....
Printer::StartDoc...........
Printer::TransportConf
ig.................................
Printer::X......................
Printer::Y......................

SECTION 247

Printer::TransportConfig

The **TransportConfig** property is a read/write property that returns or sets the configuration scheme for the *transport layer* of the Printer object. A configuration scheme consists of the communications settings needed to transmit and receive data over a *transport layer*. The communications settings do not always have to be set through the TransportConfig property. When a transport layer is selected through the Transport property of the Printer object, the default communications settings for the selected PrinterType are automatically applied.

`Visual Basic Syntax`

Set objPrinter.TransportConfig = objConfig

`Error Values`

ABERR_GETCONFIG	An error ocurred while creating the confi object.
ABERR_INVALIDCONFIG	Invalid configuration object.
ABERR_PUTCONFIG	An error ocurred while setting the config

See Also

SECTION 248

Printer::X

The **X** property is a read-only property that returns the current printer x-coordinate position. The value returned is an interger value that will be used in the next call to the PrintText , PrintBarcode , PrintMaxicode , PrintPDF417 or DrawLine methods to determine where printing will start. After one of these methods is called, the print position is automatically returned to the left-margin so that the next set of information printed is properly left-aligned. The current left-margin value can be set using the Indentation property.

The value returned by this property is measured in printer dots. A printer dot is the smallest item that a printer can print, analagous to a pixel on a video monitor. The number of dots in an inch varies between models of printers and determines the resolution of the printer in use. Consult your printer documentation for information regarding the dot-size of your printer.

```
Visual Basic Syntax
```

intValue = objPrinter.X

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::Y......................

SECTION 249

Printer::Y

The **Y** property is a read-only property that returns the current printer y-coordinate position. The value returned is an interger value that will be used in the next call to the PrintText , PrintBarcode , PrintMaxicode , PrintPDF417 or DrawLine methods to determine where printing will start. After one of these methods is called, the print position is automatically incremented so that the next set of information printed will appear on a new line in the print output. The value used to increment the print position is determined by the current setting of the PrintSize and Font properties.

The value returned by this property is measured in dots. A printer dot is the smallest item that a printer can print, analagous to a pixel on a video monitor. The number of dots in an inch varies between models of printers and determines the resolution of the printer in use. Consult your printer documentation for information regarding the dot-size of your printer.

```
Visual Basic Syntax
```

intValue = objPrinter.Y

See Also

Printer::Advance..........
Printer::Barcode...........
Printer::BarcodeName...
Printer::CancelHardwa
reRead..........................
Printer::Copies.............
Printer::DataAvailable..
Printer::DrawLine.........
Printer::EndDoc............
Printer::Font.................
Printer::FontName........
Printer::GetHardware
Data.............................
Printer::HardwareRead.
Printer::HumanReadab
le.................................
Printer::Indentation.......
Printer::IsReading.........
Printer::LabelHeight.....
Printer::LabelWidth......
Printer::OnDataAvaila
ble...............................
Printer::Orientation.......
Printer::PrintBarcode....
Printer::PrinterType......
Printer::PrintMaxicode..
Printer::PrintPDF417....
Printer::PrintRaw..........
Printer::PrintSize..........
Printer::PrintText..........
Printer::ReadMode........
Printer::Spacing............
Printer::StartDoc...........
Printer::Transport..........
Printer::TransportConf
ig.................................
Printer::X......................